MASTER EN SCIENCES INFORMATIQUES

Un langage de haut niveau pour l'exploitation de bases de données CODASYL manipulation des données

Rosseel, Brigitte

Award date:
1979

Awarding institution:
Universite de Namur

ANNEE ACADEMIQUE 1978-1979

UN LANGAGE DE HAUT NIVEAU POUR L'EXPLOITATION DE BASES DE DONNEES CODASYL : MANIPULATION DES DONNEES.

Promoteur : J.L. HAINAUT

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en Informatique.

99067

R E M E R C I E M E N T S

Je tiens à exprimer ma profonde gratitude à Jean-Luc HAINAUT pour l'intérêt porté à ce mémoire et l'aide constante qu'il m'a prodiguée lors de la rédaction. Ses nombreux conseils m'ont permis de clarifier les difficultés théoriques de cette étude et d'en réaliser une implémentation partielle.

Enfin, que toutes les personnes que je ne peux citer et qui ont participé d'une quelconque façon à ce mémoire reçoivent ici l'expression de mes plus vifs remerciements.

TABLE DES MATIERES.

+++ I N T R O D U C T I O N +++

Depuis la fin des années soixante les systèmes de base de données connaissent un essor très important.
Cette évolution est renforcée aujourd'hui par la définition de langages de haut niveau destinés à permettre aux utilisateurs un emploi plus aisé de ces systèmes.

Ce mémoire s'inscrit dans le cadre d'un travail entrepris par l'équipe "grands fichiers" des F.U.N.D.P. à Namur et dont le but était la définition d'un nouveau système de base de données : SPHINX.
Notre objectif est la définition et l'implémentation d'un langage de haut niveau de type SPHINX pour un système de gestion de base de données Codasyl.
Un système de base de données offre généralement un langage de description des données (DDL) et un langage de manipulation de celles-ci (DML). Le DML est un ensemble d'ordres qui agissent sur les données définies grâce au DDL.
De ces deux langages nous n'étudierons que celui de manipulation de données; le DDL auquel nous faisons référence au cours de l'analyse est l'objet d'un travail parallèle réalisé par M. NGYEN LAN.
Par opposition aux DDL et DML SPHINX de départ nous appelons ces nouveaux langages DDL' et DML'.
L'intérêt de créer ces DDL' et DML' pour un système codasyl réside dans le fait que un tel système répond à des normes qui ont été suivies par de nombreux systèmes commercialisés.
Le nouveau DML que nous nous proposons de créer est de type "langage de boucle" et reprend un des objectifs principaux de DML SPHINX : permettre à l'utilisateur l'écriture de programmes clairs et bien structurés. Il libère en outre l'utilisateur de contraintes délicates et fastidieuses telles que la gestion des courants, l'initialisation de paramètres implicites, etc... liées à l'écriture d'un programme DML CODASYL.

Le mémoire dont nous venons de présenter l'objet se structure en six parties.

Après avoir rappelé quelques concepts relatifs aux bases de données (chapitre 1) nous exposons dans les chapitres deux et trois les éléments d'un système CODASYL et du système SPHINX.

Ces deux systèmes sont présentés en vue de la définition du DML', c'est pourquoi cette présentation n'en constitue pas une définition complète.

Le chapitre quatre rappelle les éléments du DDL' défini par M. NGYEN LAN et expose les étapes d'analyse qui aboutissent à la définition du DML'.

Le chapitre cinq constitue la partie pratique du travail, il décrit les principes et algorithmes du compilateur DML'.

Enfin, la synthèse et les conclusions du travail font l'objet du dernier chapitre.

Chapitre I : Quelques concepts relatifs aux bases de données.

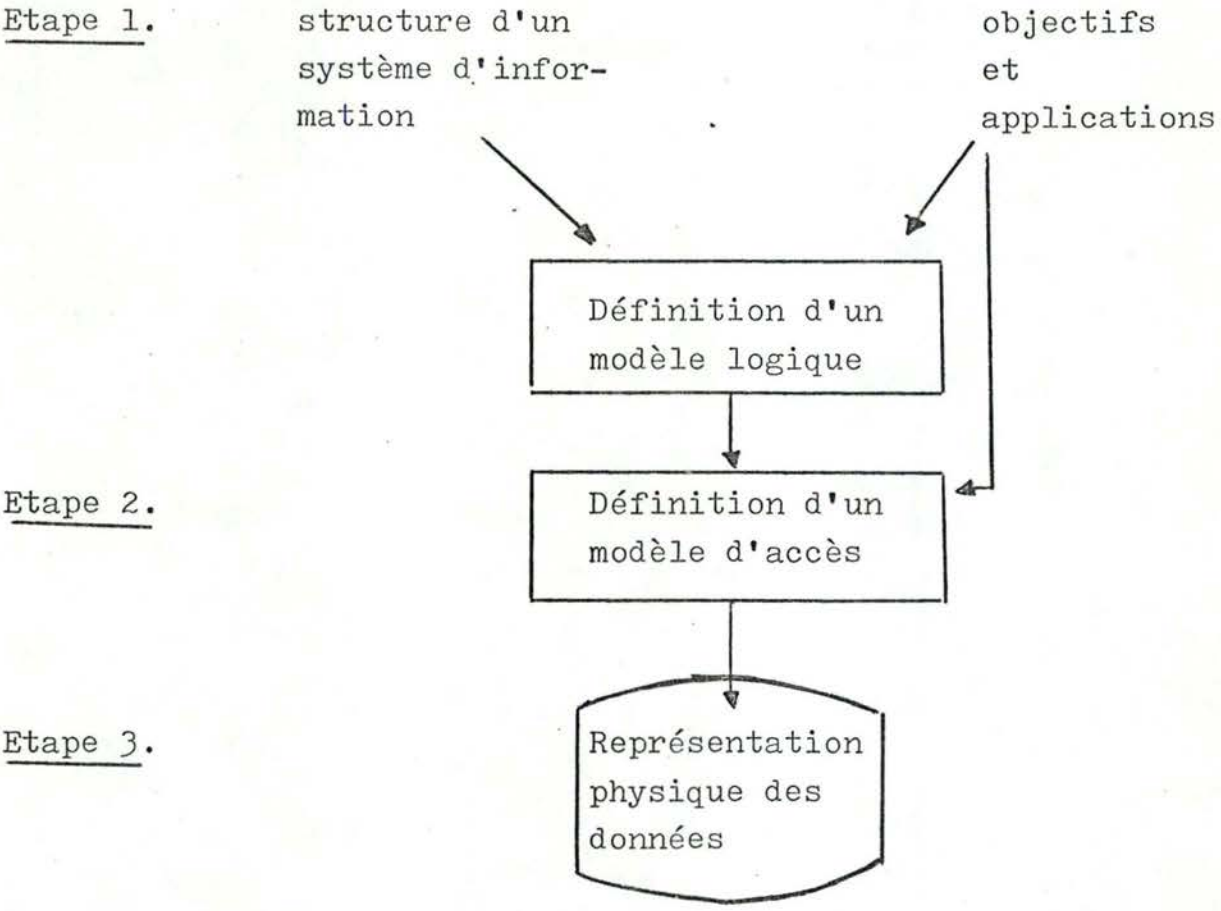
Ce chapitre a pour but d'introduire des notions générales relatives aux bases de données afin de situer le rôle et les fonctions remplis par les systèmes de base de données actuels et leurs langages associés.

Nous présentons successivement les concepts de :

1. modèles logiques, modèles d'accès et représentation physique des données.
2. de shémas externes, shémas d'accès et shémas internes.

I. 1. Modèle logique, modèle d'accès et implémentation physique.

Une base de donnée a pour but de représenter au mieux un système d'informations quel qu'il soit. Pour cela il convient d'enregistrer toutes les données du monde réel qui sont significatives par rapport aux objectifs de ce système d'informations et de les décrire en relations les unes avec les autres.
Une analyse complète d'un tel système peut se faire en trois étapes :



Dans la première étape on dégage les entités du système réel que l'on considère comme significatives ainsi que leurs propriétés et les liens qui les unissent.
On parle ainsi de structure ou modèle logique ou conceptuel.

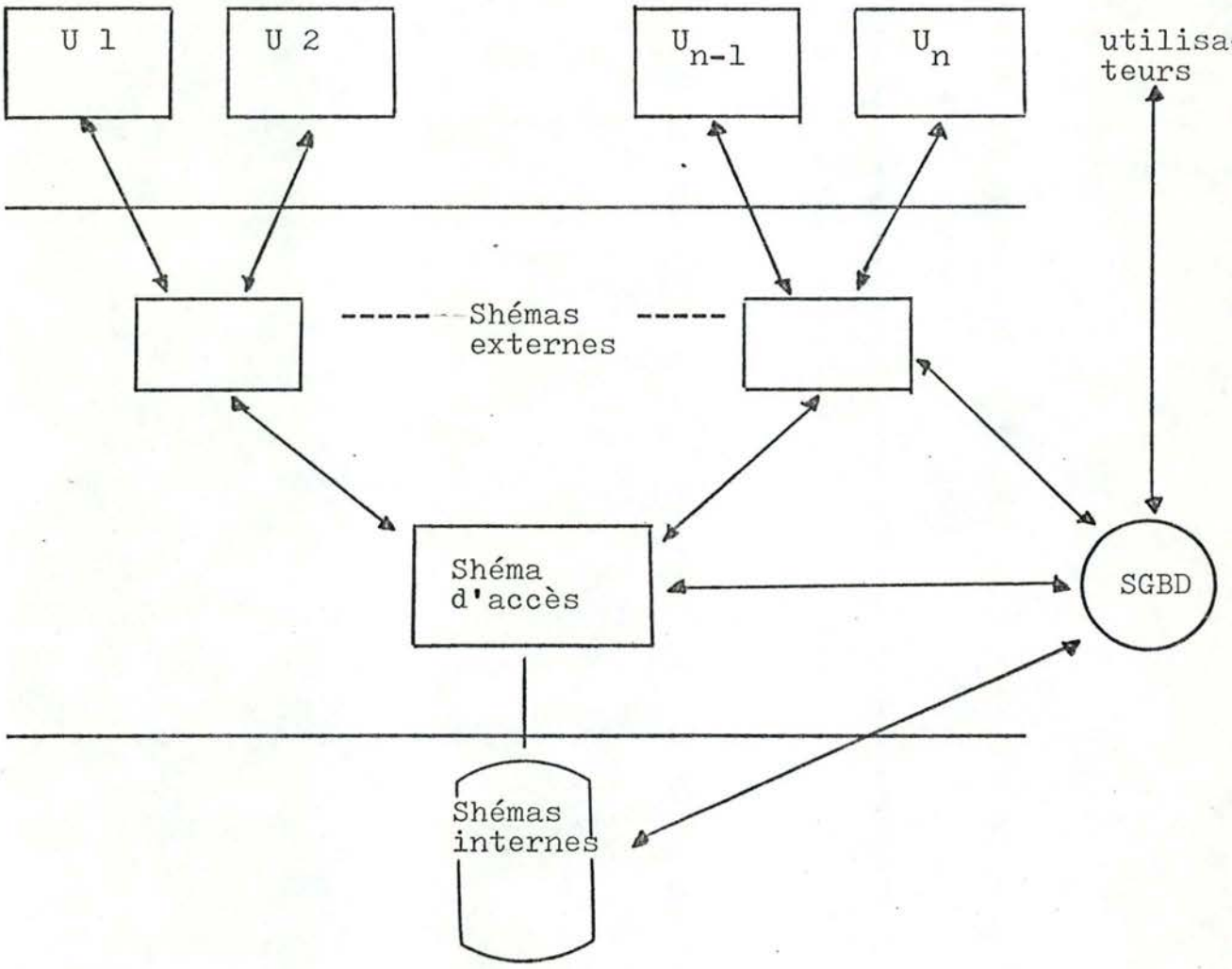
Au cours de la seconde étape on décide des accès aux différentes informations qui sont à mettre en oeuvre pour répondre au mieux aux objectifs des utilisateurs et assurer un fonctionnement efficace de la base de données. On parle dans ce cas de structure ou modèle d'accès.

Enfin dans la dernière étape on détermine quelle doit être l'implémentation physique des données et des accès définis dans les étapes précédentes.

Cette découpe en trois niveaux d'analyse permet de rendre un niveau de description indépendant des critères pris en charge dans les niveaux inférieurs.

I. 2. Shémas externes, shémas d'accès et shémas internes.

Un système de gestion de base de données (SGBD) est un ensemble d'outils permettant à un utilisateur de travailler et de considérer la base de données au niveau du modèle d'accès.
Nous pouvons représenter un tel système par la figure suivante :



- Un shéma d'accès représente la description d'une base de données au niveau du modèle d'accès.
- Un shéma externe représente la description de la même base de données mais vue sous l'angle d'un utilisateur particulier; il peut par exemple ne considérer qu'une partie du shéma d'accès et/ou en faire une description différente.
- Un shéma interne est directement lié à l'implémentation physique de la base de données.
- Le SGBD est chargé d'établir la correspondance entre modèles externes, modèles d'accès et modèles internes.
- Ces différents shémas sont construits et contrôlés par ces administrateurs de la base de données (ABD).
  Les ABD sont un ensemble de personnes chargées de définir les entités et les relations à faire entrer dans la base de données et la façon de les représenter aux différents niveaux.
- Un utilisateur communique avec le SGBD grâce aux langages de description (DDL) et de manipulations (DML) de données associées au système.
- La caractéristique principale d'un SGBD est son modèle d'accès (ou modèle de structure des données) qui détermine ses langages DDL et DML.
- Un modèle de structure de données est un ensemble de règles et de caractéristiques associées aux types de données pouvant être utilisées. Il se situe au niveau de l'implémentation logique des données.

Chapitre 2 : Le système CODASYL

- CODASYL est un comité destiné à établir des normes relatives aux spécifications de langages de programmation liés aux systèmes de données. Le DBTG (DATA BASE TASK GROUP) en est un groupe chargé d'établir des normes relatives aux langages de base de données. Plusieurs rapport ont été publiés à cet effet et de nombreux SGBD ont été développés et commercialisés sur base de ceux-ci, ce sont par ex. : DMS II00 (UNIVAC), IDMS (IBM), DBMS-20 (DIGITAL), EDMS (XEROS DATA-SYSTEMS), PHOLAS (PHILIPS), IDS (HONEYWELL).....
- Dans la suite du texte nous nous référons au rapport 71 publié par le DBT 6 et nous appelons système CODASYL un SGBD qui respecte les normes ainsi décrites.
  Un système CODASYL comprend les éléments suivants : un modèle de structure des données, un DBMS, un DDL et DML.
  Le DBMS est un ensemble d'"outils" permettant à un utilisateur d'accéder à une base de données; il communique avec le DBMS grâce aux langages DDL et DML de description et de de manipulation des données.
  Ce chapitre présente les principaux concepts d'un tel système. Rappelons que sont intérêt n'est pas d'introduire un système CODASYL mais de relever certaines notions intervenant pour l'analyse et la conception du DML'.

2.1. Le modèle de structure des données.

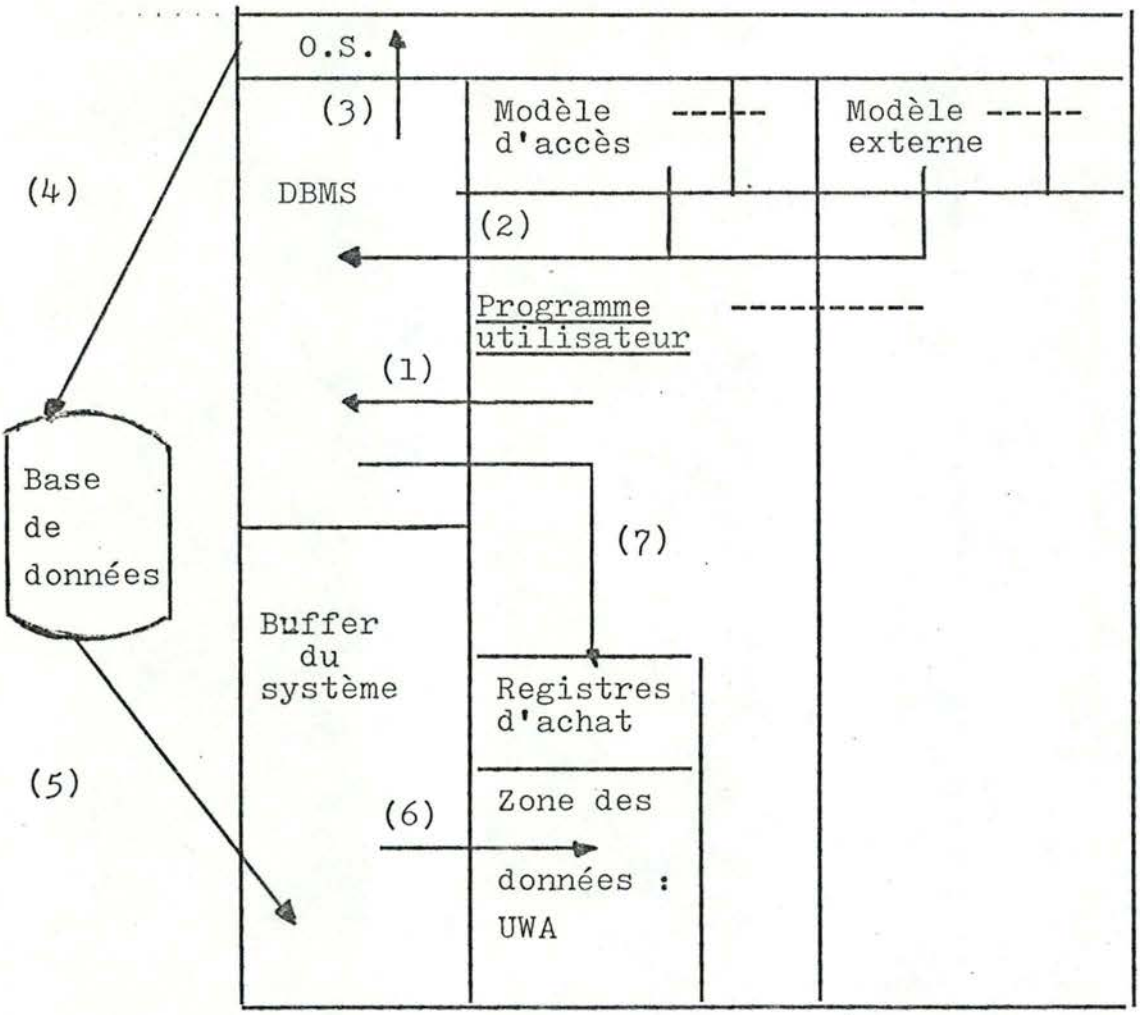
- Le système CODASYL permet de décrire une structure de données de type réseau.
  Pour le faire, il utilise les concepts de RECORD et de SET.
  Pour éviter toute confusion, nous introduirons les termes d'une part de type de record et de record et d'autre part de type de set et de set.
- Un type de record définit un ensemble d'objets possédant les mêmes caractéristiques du point de vue de l'utilisateur.
  Un record désigne un objet particulier de cet ensemble.
  Un type de record peut se décomposer en Data-items et/ou Data-aggregates.
  Un data-item représente la plus petite unité d'information et un data-aggregate est une collection nommée de data-items. (1)
  A un record correspondent des valeurs de data-items et data-aggregates.
- Un type de set définit une relation existant entre des types de record et s'exprime en termes de OWNER et MEMBER : un type de record OWNER est relié à un (ou plusieurs) type(s) de record member(s).
  A un set correspond un record OWNER et zéro, un ou plusieurs records pour chaque member.
- Un set possède les caractéristiques suivantes :
  - à partir d'un record member, on peut retrouver le record owner.
  - à partir du record owner, on peut retrouver tous les records members.
  - un record ne peut appartenir qu'à un seul set du type pour lequel son type de record est déclaré owner ou member.
  - un set est identifié par son record owner unique.
  - lors de la création d'un record, le SGBD crée automatiquement un set pour chaque type de set pour lesquels son type de record est owner, on définit ainsi un set vide : un set qui ne se compose que de son seul record owner.

---

(1) Un data-aggregate est un groupe de data-items, un vecteur ou un groupe répétitif. Un vecteur est un tableau à une dimension et le groupe répétitif est une collection nommée de données. Ces données pouvant être des vecteurs et des groupes répétitifs.

2.2. Le D B M S

Le DBMS est un intermédiaire entre d'une part le programme utilisateur et d'autre part le système (O.S.) et la base de données.
Très brièvement son fonctionnement peut se représenter par la figure suivante :



(1) Un programme utilisateur fait appel au DBMS.
Cet appel constitue une demande d'accès à la base de données.

(2) Le DBMS analyse l'appel et complète et analyse les données concernées par l'appel à l'aide du shéma d'accès et shéma externe référencé dans le programme.

(3) Le DBMS fait une demande d'I/O sur base des renseignements obtenus à partir des shémas d'accès et externes concernés par le programme.
(4) L'O.S. effectue les I/O demandés.
(5) L'O.S. transfère le résultat de cette opération dans ses buffers.
(6) Le DBMS transfère les données obtenues dans la zone des données de l'utilisateur (UWA), conformément au shéma externe concerné.
(7) Le DBMS transmet au programme des renseignements résultant de l'appel : code d'erreur,...

Il est deux notions importantes à introduire ici : les notions de DATA BASE KEY (DBK) et d'AREA.
Une AREA est une subdivision de l'espace adressable occupé par la base de données, elle peut contenir des records et des sets.
Une area est déterminée par l'administrateur de la base de données et manipulée par les utilisateurs grâce aux ordres DML OPEN et CLOSE d'ouverture et de fermeture d'area.
Lorsqu'un record est créé et placé dans une area, le DBMS lui assigne automatiquement une DBK dont la valeur constitue un identifiant pour le record et le distingue de tous les autres records de la base de données. Dans certains DBMS, cette DBK est associée à l'adresse physique d'un record dans la base de données.

Lors de l'appel d'un programme utilisateur au DBMS, il y a échange d'informations et à cet effet est prévue une zone de communication qui comprend : - une zone de données (UWA)
- des registres d'état.

2.2.1. U.W.A.(User Working Area)

Chaque programme utilisateur dispose de sa propre zone UWA qui est construite par le DBMS en accord avec le sous-shéma du programme.
A chaque type de données du sous-shéma correspond une zone de UWA possédant les mêmes caractéristiques et la même structure.
Cette zone est garnie soit par le programme utilisateur préalablement à certains ordres DML (ex : la création d'un record), soit par le DBMS suite à une demande d'accès à la base de données.

2.2.2. Les registres d'état.

On distingue : - les registres d'état courant
- les registres d'état d'erreur .

2.2.2.1. Les registres d'état courant

Ces registres existent pour toute la durée d'exécution d'un programme et lui procurent des renseignements concernant le dernier record obtenu ou créé par le programme.

Ces registres sont :

- Area-name et Record-name contenant respectivement le nom de l'area et le nom du type de record auxquels appartient le dernier record obtenu ou créé par le programme.
- Les registres CURRENTS contenant la DBK du dernier record obtenu ou créé pour : chaque type de record du sous-shéma,
  chaque type de set du sous-shéma,
  chaque area du sous-shéma,
  le programme (current du RUN-UNIT).

Ces currents sont mis à jour par le DBMS après chaque ordre DML entrainant un accès à la base de données. Ils constituent un moyen de de communication pour le programme dans le sens où un ordre DML peut référencer un record en désignant implicitement ou explicitement un current qui doit alors contenir la DBK de ce record.

L'utilisateur dispose à cet effet de plusieurs moyens de mise à jour de ces currents :

- un ordre DML constitue une demande implicite de mise à jour des currents par le DBMS.
- il peut faire une demande explicite de suppression de mise à jour de certains currents par la clause SUPPRESS CURRENCY STATUS FOR qui peut accompagner chaque ordre DML d'accès à un (ou plusieurs) record(s).
- il dispose d'un ordre DML : MOVE CURRENCY STATUS lui permettant de transférer une valeur de current dans une zone qui lui est propre.

2.2.2.2. Les registres d'état d'erreur

Ces registres indiquent si l'exécution d'un ordre DML s'est déroulée normalement ou si elle a été empêchée ou interrompue et dans les deux derniers cas quelle en est la cause.

Ces registres sont :

ERROR-STATUS : indique le type d'erreur détectée.

ERROR-AREA : indique le nom de l'area dans laquelle une erreur a été détectée.

ERROR-RECORD : indique le nom du type de record pour lequel une erreur a été détectée.

ERROR-SET : indique le nom du type de set pour lequel une erreur a été détectée.

ERROR-TYPE : indique le nom du type d'interférence avec un programme concurrent empêchant l'exécution de l'ordre.

ERROR-COUNT : indique le nombre d'erreurs détectées.

2.3. Le D D L

Le DDL est le langage dont dispose l'utilisateur pour décrire une base de données.
Cette description consiste en deux sections :
-une clause d'introduction.
-des clauses de description : - une ou plusieurs clauses AREA,
- " " " " RECORD
- " " " " SET.

2.3.1. La clause d'introduction

Cette clause permet de nommer la base de données ou partie de celle-ci. Elle s'exprime en termes de SHEMA ou SOUS-SHEMA.
Un shéma décrit la base de données complète , alors qu'un sous-shéma ne décrit qu'une partie de celle-ci.
La notion de sous-shéma permet à un utilisateur de ne travailler qu'avec la partie de la base de données qui l'intéresse. Il peut ainsi ne décrire dans la suite du texte DDL que certains éléments (type de record et set, data-items et aggregates, areas) du shéma et/ou en faire une description différente.
Le DBMS est chargé d'établir la correspondance entre shéma et sous-shéma.
La clause d'introduction peut s'accompagner d'une clause LOCK désignant les clés à fournir pour l'accès à un shéma ou sous-shéma.

2.3.2. Clauses Area

Une telle clause permet de nommer une area du shéma ou sous-shéma et d'imposer des règles d'accès à celle-ci sous forme de clés (LOCKS) réservées pour certaines opérations sur l'area.

2.3.3. Clauses Record

Une telle clause permet de nommer et de décrire un type de record.
Cette description comporte trois sections:
- une clause WITHIN et AREA-ID éventuelle permettant de localiser l'area dans laquelle se trouve un record du type décrit.

- une clause LOCATION MODE indiquant comment est localisé un record de ce type dans la base de donnée.
- des clauses décrivant la structure du type de record: elles définissent les data-items et/ou data aggregates qui le compose.

Clause WITHIN

Elle indique dans quelle(s) area(s) peuvent se retrouver les records du type de record concerné.
Un type de record peut appartenir à plusieurs areas mais un record n'appartient qu'à une seule.

Clause AREA-ID

Cette clause accompagne une clause WITHIN lorsque celle-ci comporte plus d'un nom d'area.
Elle a pour effet de réserver une zone en UWA destinée à contenir un nom d'area.
Cette zone est garnie par le programme, préalablement à certains ordres DML impliquant une clause LOCATION MODE CALC (cfr. ci-après), elle permet alors de préciser dans quelle area de la clause WITHIN se trouve le record concerné par l'ordre.

Clause LOCATION MODE

Elle indique comment de DBMS localise un record du type concerné par cette clause et par conséquent les renseignements que devra fournir le programme pour identifier ce record.
Cette clause joue un double rôle :
- elle intervient lors de la création d'un record; elle indique alors avec la clause WITHIN ou AREA-ID comment et où sera inséré le record dans la base de données.
- elle intervient pour localiser un record OWNER d'un set s'insérant dans un chemin d'accès défini par une SOS (cfr. 2.3.4.3.).

Il existe trois modes de localisation d'un record dans une area. Ces modes sont décrits par les clauses suivantes :

LOCATION MODE DIRECT Z 1

La localisation est basée sur la valeur de Z 1 qui doit contenir une DBK. Z 1 doit donc, préalablement à un ordre DML, être garnie par le programme avec la DBK voulue.

LOCATION MODE CALC USING <data-item 1> , <data-item 2> ,...
DUPLICATES | NOT | ALLOWED :

la localisation est alors basée sur les valeurs des data-items concernés qui par ce fait sont déclarés CALC KEYS.
La DBK affectée au record est fonction de ces valeurs et du nom de l'area fournie soit par la clause within, soit par area-id. La clause DUPLICATES indique si les valeurs de calc keys sont uniques ou non.

LOCATION MODE VIA ⟨nom type de set⟩ SET
La localisation est alors basée sur l'appartenance du record à un set (de type spécifié dans cette clause) en tant que member. Le record sera placé le plus près possible de son point d'insertion logique dans le set.
Le DBMS procède à la localisation en deux étapes :

- il localise le set grâce à la SOS associée au type de set (cfr. 2.3.4.3.)
- parmi les record members, il sélectionne celui désiré.

Le programme doit donc fournir, préalablement à un ordre DML, le moyen de localiser le set concerné et au sein de celui-ci le member.

Structure d'un type de record.
La description d'un type de record se fait par niveaux comme en Cobol, chacun des niveaux décrit un data-item ou data-aggregate.
La description d'un data-item se fait par des clauses PICTURE, TYPE, OCCURS comme en Cobol et peut en outre s'accompagner des clauses suivantes :

{ACTUAL / VIRTUAL} RESULT OF ⟨nom procedure⟩ .
Cette clause indique que la valeur du data-item décrit résulte de l'exécution d'une procédure citée dans cette clause; le terme ACTUAL indique que la valeur existe physiquement dans la base de données pour le record; VIRTUAL est utilisé dans le cas contraire.

{ACTUAL / VIRTUAL} SOURCE IS ⟨nom data-item⟩ OF ⟨nom type de set⟩ .
Les termes actual et virtual prennent le même sens que ci-dessus.
Cette clause indique que la valeur du data-item est redondante avec celle du data-item de l'owner du record concerné pour un type de set spécifié.

CHECK
Cette clause indique qu'un contrôle sera effectué sur la valeur de ce data-item lors d'une insertion ou modification de celle-ci dans la base de données.

2.3.4 Clauses set.

Une telle clause permet de nommer et de décrire un type de set du shéma ou sous-shéma.
Cette description comporte trois sections :
- des clauses concernant l'implémentation physique d'une relation dans la base de données : mode de chaînage,.... Nous n'examinons pas ces clauses car elles n'interviennent pas dans la suite de l'analyse.
- des clauses identifiant le type de ste en termes d'OWNER et MEMBERS.
- une clause d'identification d'un set: SET OCCURENCE SELECTION (SOS).

2.3.4.1 Clause owner.

OWNER IS { < nom de type record > .
         { SYSTEM .

Lorsque l'OWNER est system, on définit un type de set particulier qui ne comporte qu'un seul set et dont le record OWNER est appelé system.

2.3.4.2. Clause Member

MEMBER IS < nom type de record> { MANDATORY / OPTIONAL } { AUTOMATIC / MANUAL }

Un type de record est member mandatory pour un type de set si lorsqu' un record de ce type est inséré dans un set de ce type, il ne peut plus en être détaché pendant toute sa durée de vie dans la base de données; il est MEMBER OPTIONAL sinon.
Un type de record est MEMBER AUTOMATIC pour un type de set, si lors de la création d'un record de ce type, son insertion dans un set est automatiquement prise en charge par le DBMS; il est MEMBER MANUAL sinon.

Outre cette clause, l'identification d'un type de set peut s'accompagner d'autres clauses optionnelles :
- DUPLICATES NOT ALLOWED FOR <nom data-item 1>,
  <nom data-item 2>,----

  Cette clause indique que des records members ne peuvent avoir les mêmes valeurs pour les data-items spécifiés.
- ASCENDING / DESCENDING KEY <nom data-item 1>, <nom data-item 2>...
  [DUPLICATES [NOT] ALLOWED]

  Cette clause permet de définir un ordre (croissant ou décroissant)

d'insertion d'un member dans un set sur base des valeurs des data-items spécifiés.
La clause DUPLICATES indique si ces valeurs sont uniques ou non pour un RECORD MEMBER.

- SEARCH KEY <nom data-item 1> , <nom data-item 2> ,...
  [DUPLICATES [NOT] ALLOWED]

  Cette clause indique que pour chaque set, un index sera créé, permettant un accès direct à chacun des records members sur base de leurs valeurs pour chacun des data-items spécifiés. La clause duplicates indique si ces valeurs sont uniques ou non pour un record.

2.3.4.3. Set Occurence Selection (SOS)

Cette clause est la déclaration du mode de sélection automatique d'un set. Plusieurs ordres DML exigent à leur exécution, la spécification d'un (ou plusieurs) set(s) susceptible(s) d'intervenir dans l'opération. Un set est spécifié par son record owner unique. La déclaration du mode de sélection automatique consiste à donner au SGBD la description d'un chemin d'accès. Ce chemin est une chaîne de sets à parcourir dans le sens owner vers member.
Le point de départ doit être repérable soit par la valeur du current de son type de set soit par la location mode de l'owner qui doit être DIRECT ou CALC.
Le point d'arrivée de ce chemin est un set du type auquel la SOS est relative. Un chemin est de longueur nulle lorsque le point de départ égale le point d'arrivée.
Les étapes du chemin autres que le point de départ consistent chacune à sélectionner un set par identification de son owner comme member du set sélectionné à l'étape précédente.
Le rapport CODASYL propose deux méthodes de description de ce chemin :

- la première consiste à le décrire explicitement depuis le point de départ jusqu'au point d'arrivée.
- la deuxième consiste à décrire le chemin à partir du point d'arrivée et dans ce cas les étapes du chemin se font par références successives à des clauses LOCATION MODE et SOS.

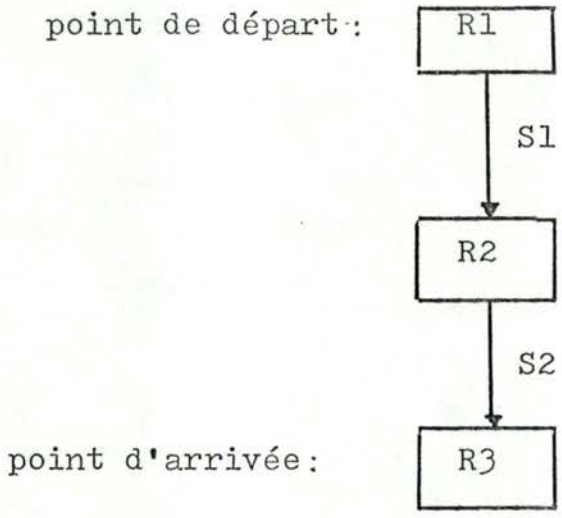
L'identification d'un member dans un set se fait par valeurs de certains data-items déclarés identifiants de set, c-à-d : les data-items déclarés dans une clause DUPLICATES NOT ALLOWED FOR
ASCENDING / DESCENDING KEY IS... DUPLICATES NOT ALLOWED
SEARCH KEY IS ... DUPLICATES NOT ALLOWED

Exemples

lère méthode

Soit les clauses : RECORD R1 LOCATION MODE IS DIRECT Z1
RECORD R2
RECORD R3
SET S1 OWNER IS R1
MEMBER IS R2 Duplicates not allowed
for d-it 1, d-it 2.
SET S2 OWNER IS R2
MEMBER IS R3
SOS THRU S1 USING Location mode of owner
S2 USING d-it 1, d-it 2.

La SOS associée au type set S2 définit un chemin de longueur deux :



Un set de type S2 sera identifié comme suit :

- on identifie un Record R1 par son location mode (Z1 doit contenir sa DBK) et de ce fait on identifie un set de type S1.
- dans ce set, on identifie un member de type R2 grâce à ses valeurs d'identifiants de set : d-it 1, d-it 2 (les valeurs doivent être présentes en UWA) et de ce fait un set de type S2 est localisé, le record de type R2 sélectionné désigne l'owner.

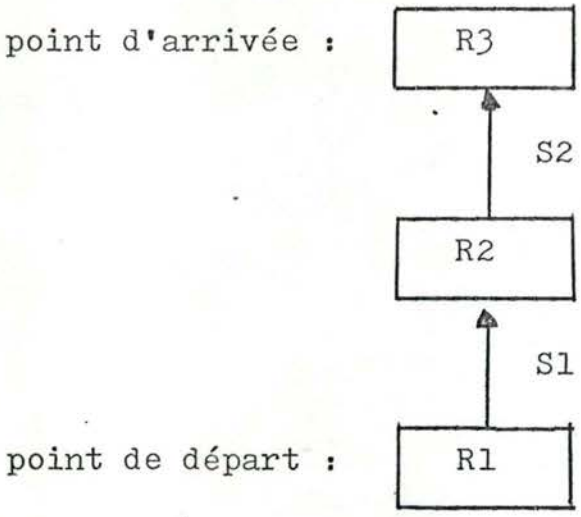
2ème méthode

Soit les clauses DDL :

RECORD R1
RECORD R2 LOCATION MODE IS VIA S1
SET S1 OWNER IS R1

```
                MEMBER IS R2 DUPLICATES NOT ALLOWED FOR
                                    d-it 1, d-it 2
                SOS THRU CURRENT OF SET
SET S2          OWNER IS R2
                MEMBER IS R3
                SOS THRU LOCATION MODE OF OWNER USING d-it 1,
                                                      d-it 2.
```

La SOS associée au type de set S2 définit un chemin de longueur deux :



Un set de type S2 sera identifié comme suit :

- on identifiera le record de type R2 owner grâce à son location mode, qui renvoie à la SOS du type de set S1.
  Le record de type R2 sera identifié en tant que member d'un set de type S1 grâce à ses valeurs d'identifiant de set : d-it 1 et d-it 2
- un set de type S1 est identifié par le current de ce type de set qui contiendra la DBK de l'owner.

2.4. Le D M L

Le DML est le langage utilisé par un programme utilisateur pour communiquer avec le DBMS dans le but d'accéder et de manipuler des informations de la base de données.
Le DML n'est pas un langage à part entière, il s'insère dans un autre (langage hôte) qui est généralement le COBOL.
Les principaux ordres DML sont :

- INVOKE

Forme générale :

INVOKE SUB-SHEMA ⟨nom d'un sous-shéma⟩ OF SHEMA ⟨nom d'un shéma⟩

Cet ordre fait référence à un sous-shéma particulier. Il ne peut exister qu'un seul ordre INVOKE dans un programme et par conséquent un utilisateur ne peut travailler qu'avec un seul sous-shéma dans un programme.

OPEN

Forme générale :

- OPEN { FOR SET ⟨nom type de set⟩ ... / AREA ⟨nom area⟩ .... }
[USAGE MODE IS [EXCLUSIVE / PROTECTED] { RETRIEVAL / UPDATE }]

Cet ordre permet l'ouverture des areas relatives aux types de set cités (FOR SET) ou explicitement nommées (AREA).
La clause USAGE MODE détermine les règles de concurrence associées à ces areas :
si la clause est :

EXCLUSIVE : le programme se réserve l'accès exclusif aux données des areas concernées.

PROTECTED : le programme se réserve le droit de mise à jour des données des areas concernées.

UPDATE : cette clause permet à d'autres programmes d'ouvrir les mêmes areas avec un usage mode autre que exclusive ou protected.

RETRIEVAL : cette clause permet à d'autres programmes d'ouvrir les mêmes areas avec un usage mode autre que protected.

Par défaut, l'usage mode est RETRIEVAL.

- FIND

Forme générale :

FIND ⟨expression de sélection⟩

[SUPPRESS | ALL | RECORD | AREA | SET | nom type de set , ... | CURRENCY UPDATES]

Cet ordre constitue une demande d'accès à un record de la base de données.

- La clause ⟨expression de sélection⟩ indique comment sera identifié le record à obtenir.
  Cette sélection se fait selon trois critères :
  - on peut accéder à un recor en fournissant directement sa DBK (format 1).
  - on peut accéder à un record par référence à un current (format 2,3,4).
  - on peut accéder à un record en donnant la valeur de certains de ses data-items (format 5,6,7).
- La clause SUPPRESS constitue une demande de non-mise à jour de certains currents, spécifier dans cette clause; si c'est ALL seul le current du run-unit sera mis à jour.

- GET

GET | [⟨nom type record⟩]. | ⟨nom type de record⟩ ; ⟨data - item⟩ .... |

Cet ordre fait suite à un ordre FIND et met à la disposition du programme (en UWA) la valeur des data-items spécifiés et appartenant au record spécifié par le current du run-unit.

- STORE

STORE ⟨nom type de record⟩ Suppress ....

Cet ordre créé un nouveau record du type spécifié. Les valeurs de data-items associées à ce record se trouvent en UWA; de plus, le record est inséré dans un set pour lequel il est AUTOMATIC MEMBER, chacun de ces sets est identifié par SOS.

- INSERT

INSERT [<nom type de record>] INTO { <nom type de sets>.... / ALL SETS }

Cet ordre insère un record du type spécifié dans chaque set de type spécifié et identifié par current.
Le record à insérer doit être optional automatic, optional manual ou mandatory manual member pour chacun des sets.

- MODIFY

MODIFY { [<nom type de record>] / <nom type de record> ; data-item 1,... } USING <data-item 2>

Cet ordre modifie les valeurs de data-items d'un record de type spécifié ou l'appartenance de ce record à un type de set (avec clause USING) et dans ce cas <data-item 2> .... sont des data-items dont les valeurs (en UWA) permettent de reconstituer la SOS du nouveau set.

- CLOSE

CLOSE { ALL [FOR SET nom type de set ,...] / AREA <nom area> ,... }

Cet ordre permet la fermeture des areas spécifiées.

- DELETE

DELETE [<nom type de record>] [ONLY / SELECTIVE / ALL]

Cet ordre supprime un record de la base de données.
Si il s'accompagne de la clause :
ONLY : tous les records mandatory member d'un set dont il est owner sont également supprimés et cette règle s'applique également à ces records.
SELECTIVE : la même règle s'applique aux records mandatory members et optional members s'ils n'appartiennent pas à d'autres sets.
ALL : tous les records member sont supprimés et cette règle s'applique pour ces records.
Lorsqu'aucune de ces trois clauses n'est présente, l'ordre s'exécute pour le record identifié par current du run-unit uniquement s'il n'est owner que de sets vides.

- REMOVE

REMOVE ⟨nom type de record⟩ FROM { ⟨nom type sets⟩ ... / ALLS SETS }

Cet ordre détache un record des sets de type spécifiés et pour lesquels il est optional MEMBER.

- MOVE

Forme 1

MOVE CURRENCY STATUS FOR { RUN-UNIT / ⟨nom type de record⟩ RECORD / ⟨nom d'area⟩ AREA / ⟨nom type de set⟩ SET } TO ⟨nom zone Cobol⟩

Cet ordre sauve une valeur de current dans une zone du programme.

Forme 2

MOVE AREA-NAME FOR { RUN UNIT / ⟨nom type de record⟩ RECORD / ⟨nom d'area⟩ AREA / ⟨nom type de set⟩ SET / ⟨nom zone-1 Cobol⟩ } TO ⟨nom zone-2 Cobol⟩

Cet ordre sauve dans une zone du programme la valeur de AREA-NAME contenant le nom d'area du record identifié par current ou dont la DBK se trouve dans une zone du programme.

KEEP

Cet ordre permet à un programme de mettre un record sous surveillance du DBMS. Lors d'une nouvelle utilisation de ce record dans un ordre DM le DBMS préviendra (grâce aux registres ERROR-STATUS et ERROR-TYPE) si le record a été "modifié" par un autre programme.

- FREE

Cet ordre annule l'effet d'un ordre KEEP.

2.5. La concurrence entre programmes.

Des programmes sont dits concurrents lorsqu'ils utilisent simultanement les mêmes données.
Cette concurrence peut avoir des repercutions multiples sur les données : une modification peut les rendre invalides au niveau global de la base de données et/ou au niveau d'un programme concurrent.
Etant donné le cadre dans lequel se situe ce travail, nous ne considerons que ce deuxième aspect de la concurrence.
Quelques exemples permettent d'illustrer les inconvénients liés à cette notion.

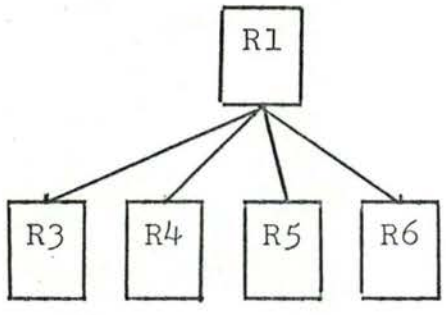
Exemple 1 :

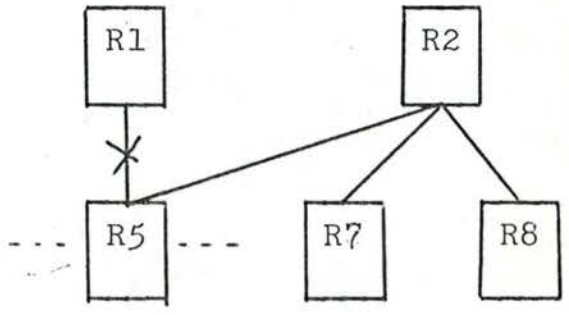
programme 1	programme 2
FIND REC USING data-item1 , data-item2 GET REC.	MODIFY REC ; data-item2

Si l'ordre de modification s'exécute avant l'ordre GET, le progamme 1 éxécutera cet ordre pour un record ne répondant plus au critère de sélection défini par le FIND.

Exemple 2 :

programme 1	programme 2
ce programme examine tous les records members d'un set	exécute la modification suivante:





Si l'ordre de transfert de R5 dans un autre set de même type s'exécute en même temps que l'examen de R5 dans le programme 1, celui-ci continuera à examiner le set à partir de R5 et par conséquent accèdera à R7 et R8 plutôt qu'à R6.
Il apparaît donc important d'offrir à l'utilisateur le moyen d'assurer l'usage exclusif des données dans de tels cas.
Le système CODASYL est limité à ce sujet: il permet l'imposition de règles de concurrence au niveau d'une area grâce à la clause USAGE MODE; par conséquent un utilisateur désirant avoir l'usage exclusif de certaines données bloque également toutes celles appartenant à la même area.
Les ordres KEEP et FREE permettent également un certain contrôle dans le sens où on peut vérifier pour chaque ordre DML si les données concernées et ayant fait l'objet d'un ordre KEEP ont été modifiées depuis leur dernière utilisation.
Remarquons que ces deux ordres ne sont implémentés pour aucun DBMS de type CODASYL, c'est pourquoi nous n'en tiendrons pas compte dans la suite du texte.

Chapitre 3 : Le système sphinx

Le système sphinx résulte de recherches entreprises par l'équipe "grands fichiers" des F.U.N.D.P. à Namur. Il définit un modèle conceptuel (appelé MSI), un modèle d'accès, des langages de description (DDL), de manipulation (DML) et d'interrogation (NUL) des données.
Le système choisi pour la réalisation de SPHINX est le système SESAM de SIEMENS.
Ce chapitre étudie le système SPHINX dans ses principes et la présentation est essentiellement centrée sur le DML.

3.1. Le modèle d'accès

Ce modèle est de type modèle relationnel binaire.
Les éléments et concepts sont les suivants :

- Les informations présentes dans la base de données sont regroupées en classes appelées OBJET. Une information particulière d'une classe est appelée réalisation d'un objet.
- Les objets dont les réalisations sont des valeurs sont dits élémentaires (O.E.) sinon ils sont dits complexes (O.C.).
  (Les notions d'O.E. et d'O.C. se rapprochent des notions de data-item et type de record CODASYL). Il existe toujours un objet particulier, la RACINE, dont l'unique réalisation représente la base de données.
- Un objet élémentaire est dit composé si chaque réalisation peut être décomposée en éléments qui sont eux-mêmes des réalisations d'objet élémentaires.
- Une relation d'un objet (appelé ORIGINE) vers un autre objet (appelé CIBLE) correspond à un chemin d'accès qui à partir de chaque réalisation origine permet d'accéder à un certain nombre de réalisations cible.
- Une relation peut être définie entre deux objets complexes, d'un objet complexe vers un objet élémentaire, d'un objet élémentaire vers un objet complexe et de la racine vers un objet complexe.
- Deux relations peuvent être déclarés inverse l'une de l'autre.
- Une relation est caractérisée par 4 paramètres I-J, K-L qui indiquent que :
  - à partir d'une réalisation origine, la relation permet d'accéder à un nombre de réalisations cibles compris entre I et J.
  - la réalisation permet d'accéder à toute réalisation cible à partir d'un nombre de réalisations origines compris entre K et L.

3.2. Communication entre programme et DBMS

Le DBMS permet à un programme DML-SPHINX de communiquer avec la base de données.

La zone de communication se compose :

- de variables indiquant par leurs valeurs la manière dont s'est déroulé un ordre DML.
- d'une zone de données destinées à contenir les informations échangées :
  lors de l'accès à des réalisations d'O.C., un programme peut spécifier les O.E. liés à cet O.C. dont il désire obtenir les valeurs, ces valeurs sont alors rangées dans des zones dont le nom est celui de l'O.E. d'origine et dont la découpe correspond à celle décrite dans le DDL.

3.3. Le principe de la boucle d'accès

La boucle d'accès permet d'effectuer des traitements de type répétitif.
Une manière quelque peu formelle d'exprimer les opérations réalisées serait : pour chaque ⟨nom d'un type d'objet⟩ faire :
⟨séquence d'instruction⟩
dont l'interprétation serait : accéder à toutes les réalisations du type d'objet spécifié et pour chacune d'elle exécuter la séquence d'instruction.
Dans une boucle contrôlée par des valeurs successives d'une variable, on parle généralement de valeur courante de la variable de boucle; dans le contexte d'un système de base de données SPHINX, on introduit la notion de réalisation courante de la boucle.
De manière à pouvoir désigner cette réalisation dans le corps de la boucle, on lui affecte une étiquette qui correspond au nom de la variable de boucle.

3.3.1. Boucle et bloc de base

Ces notions interviennent fréquemment dans la suite du texte, c'est pourquoi il est nécessaire de les définir préalablement.
La boucle de base d'un ordre DML est la boucle d'accès la plus imbriquée dans laquelle se trouve cet ordre.
Le bloc de base d'un ordre DML généralise la notion de boucle de base à tous les ordres DML définissant une structure de bloc:
le DML définit avec les ordres d'accès (OPEN et REACH) un autre ordre (PREPARE) définissant une structure de bloc.

3.4. Les ordres du DML

3.4.1. Ordres définissant une boucle d'accès

3.4.1.1. Boucle d'accès à une base de données

Avant d'accéder aux réalisations d'un objet, il est nécessaire d'accéder à la base de données sur laquelle on désire travailler.
La spécification de cet accès prend la forme d'une boucle d'accès à l'unique réalisation de l'objet racine qu'est la base de données, et à cette boucle ne correspond qu'une seule itération.
Les ordres de cette boucle sont : OPEN et CLOSE.
Un programme DML peut ouvrir plusieurs bases de données.
A un bloc OPEN-CLOSE est associée une étiquette qui désigne la base de données courante de ce bloc.
Exemple : supposons l'existence des bases de données STOCK et FOURNISSEUR :

```
@ OPEN STOCK = STK-1 .
          OPEN FOURNISSEUR = F-1 .
@ CLOSE STK-1 .
```

Remarquons que l'ordre CLOSE relatif à FOURNISSEUR est implicite dans l'ordre CLOSE STK-1.

3.4.1.2. Boucle d'accès à des réalisations d'un objet complexe

3.4.1.2.1. Structure de la boucle d'accès

L'unique moyen d'accéder aux réalisations d'un O.C. est d'emprunter un chemin d'accès qui s'exprime sous la forme d'une relation dont l'O.C. est cible (cfr. modèle d'accès).
Cet accès est demandé au moyen d'une boucle d'accès dont les ordres sont : REACH et END.
Un accès étant réalisé à partir d'une réalisation origine, l'ordre REACH doit spécifier explicitement ou non :
- un objet complexe dont on désire obtenir les réalisations,
- une relation d'accès dont cet O.C. est cible,
- une réalisation de l'objet origine de cette relation.

Exemple : supposons une relation EMPLOYE dont l'objet origine est DEPARTEMENT et l'objet cible PERSONNE.
L'accès aux réalisations des personnes à partir de cette relation se fera comme suit :

@ REACH PERSONNE = P1 FROM DEPT VIA EMPLOYE
<traitement sur P1>
@ END

Ces ordres impliquent que :
le bloc P1 est inclus dans un bloc DEPT qui est une boucle d'accès à des réalisations de DEPARTEMENT, c-à-d que l'étiquette DEPT est connue à l'endroit de l'ordre REACH P1.
Il est permis d'omettre la clause FROM, cette omission désigne implicitement l'ordre correspondant au bloc de base de l'ordre REACH.
La réalisation courante d'une boucle est connue sous le nom de l'étiquette définie dans l'ordre REACH. Cette étiquette devient sans signification et inconnue à l'extérieur du bloc auquel elle est attachée (c-à-d après l'ordre END correspondant).
Le corps d'une boucle est exécuté pour chaque réalisation courante de la boucle, il en résulte que si il n'existe pas d'occurences de relation effectives entre les objets cibles et origines concernés par l'ordre REACH, la boucle ne sera pas exécutée et l'exécution du programme se poursuit à l'instruction suivant directement l'ordre END.

3.4.1.2.2. Accès soumis à des conditions

Il arrive qu'on ne désire obtenir que certaines des réalisations auquelles la réalisation permet d'accéder. Cette restriction d'accès s'exprime dans un ordre REACH par une clause IF qui permet de définir les conditions qui doivent vérifier les réalisations pour être retenues. Ces conditions sont donc limitées à celles qui portent sur les valeurs d'O.E. liés aux réalisations de l'O.C.
Ces conditions s'expriment sous forme d'une expression booléenne et les opérateurs de comparaison d'une condition sont d'une part ceux d'usage dans une expression conditionnelle en Cobol et d'autre part des opérateurs d'appartenance ou non à une liste de valeurs.
Exemple : Supposons dans l'exemple précédant un accès sélectif aux employés habitant NAMUR ou BRUXELLES et âgés de moins de 30 ans :

@ REACH PERSONNE = P1 VIA EMPLOYE IF (LOCALITE = 'NAMUR',
'BRUXELLES'
AND AGE < 30).

3.4.1.3. Accès aux réalisations d'objets élémentaires

L'acquisition de valeurs d'O.E. est directement liée à l'ordre REACH d'accès aux réalisations de l'O.C. auxquelles elles sont reliées.
Cet accès se fera au moyen d'une clause GET dans un ordre REACH; elle constitue soit une demande globale (GET ALL) de toutes les valeurs d'O.E. attachées à une réalisation d'O.C. soit une demande sélective (GET <liste d'O.E.>) de certaines de ces valeurs.
L'absence de clause GET indique que l'ordre REACH ne désire pas obtenir les valeurs d'O.E. liées aux réalisations d'O.C. concernées par l'ordre.
Lors de l'exécution d'un ordre REACH, les valeurs demandées seront rangées dans les zones Cobol dont le nom est celui de l'O.E. correspondant.

3.4.2. Contrôle des boucles d'accès

La boucle REACH-END constitue un moyen d'obtenir par une relation toutes les conditions d'un objet complexe vérifiant éventuellement des conditions et d'effectuer pour chacune la totalité des instructions de corps de la boucle.
De nombreux problèmes exigent que pour certaines réalisations on n'exécute pas l'entièreté du corps de la boucle, mais que l'on passe prématurément à la réalisation suivante.
Dans d'autres problèmes on désire dans certaines conditions terminer complètement l'exécution d'une boucle en abandonnant toutes les réalisations auxquelles la boucle a donné accès.
Afin de répondre à ces deux types de problème, le DML-SPHINX comporte les ordres NEXT et EXIT.

3.4.2.1. NEXT

La forme générale de cet ordre est : @NEXT [<étiquette>],
où <étiquette> est le nom d'une étiquette active à cet endroit, de la même manière que dans les autres verbes, l'absence de la clause clause <étiquette> indique que l'on désigne implicitement l'étiquette du bloc de base.
Cetordre commande l'abandon du traitement de la réalisation courante correspondant à l'étiquette citée et le passage à la réalisation suivante.

Si la boucle ⟨étiquette⟩ n'est pas le bloc de base de l'ordre NEXT, tous les blocs inclus dans cette boucle et contenant cet ordre sont abandonnés avant le passage à la réalisation suivante.
Un ordre NEXT n'a de sens que pour une boucle REACH-END.

3.4.2.2. EXIT

La forme générale de cet ordre est : @ EXIT [⟨étiquette⟩],
où ⟨étiquette⟩ prend le même sens que pour l'ordre NEXT.
Cet ordre commande l'abandon de la boucle et la sortie du contexte; l'exécution se poursuit à la première instruction qui suit l'ordre de cloture de la boucle que l'on quitte.

3.4.3. Variables de travail

Cette notion est née d'un inconvénient majeur lié à la structure de blocs : l'impossibilité de travailler avec une réalisation courante hors du contexte qui la définit.
Le cas typique illustrant cet inconvénient est celui d'un programme désirant effectuer un même traitement sur une réalisation d'un O.C. pouvant être obtenue par des voies différentes et par conséquent par des boucles d'accès différentes.
Compte tenu de la structure de bloc, on est obligé dans un tel cas d'écrire le traitement autant de fois qu'il existe de moyens d'accès à une réalisation de l'O.C.
L'introduction de variables de travail permet de lever cette obligation.
Une variable de travail (VT) est une variable DML, vide au départ, qui peut contenir la référence à une réalisation d'un objet complexe dans la base de données.
Une variable de travail est connue dans tout le programme et sa valeur et son utilisation ne sont donc pas soumises aux contraintes de la structure de bloc.
Des ordres DML peuvent affecter une valeur à une VT, libérer et tester le contenu d'une VT et il existe un ordre d'accès à une réalisation d'O.C. référencée par une VT.

De ces ordres nous ne retenons que les suivants :

SAVE

Forme générale :

@ SAVE <nom étiquette> IN <variable de travail>

Cet ordre affecte à une variable de travail la référence à la réalisation d'O.C. à laquelle est affectée l'étiquette nommée.

REACH (Format 2)

Forme générale :

REACH <nom d'O.C.> <variable de travail> = <étiquette>..

Cet ordre accède à la réalisation de l'objet complexe nommé dont la référence se trouve dans la variable de travail citée.

3.4.4. Les ordres de modification

3.4.4.1. Ordre de création d'un objet complexe : CREATE

Lorsqu'un O.C. est cible d'une réalisation telle que K > 0 ou I > 0, la création d'une de ses réalisations ne peut s'effectuer que si l'on a spécifié pour chacune des relations : K origines et I cibles, auxquelles cette réalisation devra être attachée dès son insertion dans la base de données.

Exemple :

Supposons : la relation EMPLOYE (DEPARTEMENT PERSONNE)
dont les paramètres sont : I = 0 et J = ∞
K = 1 et L = 1
et la relation BUREAU (PERSONNE, LOCAL)
dont les paramètres sont : I = 1 et J = 1
K = 0 et L = 1

La création d'une réalisation d'un O.C. PERSONNE nécessitera outre cette réalisation : - une réalisation de l'O.C. DEPARTEMENT
- une réalisation de l'O.C. LOCAL.

Compte tenu des principes de base du DML et du COBOL ces spécifications se feront :

- par la citation d'étiquettes pour DEPARTEMENT et LOCAL
- par le rangement dans une zone de communication des valeurs des O.E. associées à l'objet PERSONNE à créer.

Cette zone de communication devra donc, préalablement à l'ordre de création, être réservée et garnie avec les valeurs adéquates.

Cette "préparation" de la réalisation de l'O.C. à créer se fera grâce à l'ordre PREPARE.

Forme générale de l'ordre :

@ PREPARE PERSONNE = P1 [FROM...] WITH (NO, NOM, ADRESSE) :

Cette commande permet de réserver et préparer une zone qui pourra contenir des valeurs des O.E. spécifiés dans la clauses WITH.
La clause FROM doit spécifier une étiquette d'un ordre OPEN.
L'ordre PREPARE définit un contexte, clôturé par un END, dans lequel sont étiquette est connue.
A ce stade, tout est prêt en ce qui concerne la zone de communication. Il reste encore à spécifier les objets cibles et origines des relations décrites ci-dessus. Ces spécifications se font par l'ordre CREATE, qui prend la forme suivante :

@ CREATE ⟨étiq⟩ [, ⟨des. relation⟩ : ⟨liste étiquettes⟩]$_{1}^{n}$

où ⟨étiq⟩ est l'étiquette définie par l'ordre PREPARE.
⟨listes étiquettes⟩ sont les I cibles (ou K origines) de la relation pour laquelle l'objet à créer sera origine (ou cible).

3.4.4.2. Suppression d'une réalisation d'un O.C. : SUPPRESS

La forme générale de cet ordre est :

@ SUPPRESS ⟨étiquette⟩ { EXIT . / NEXT . }

Après cet ordre, la réalisation attachée à ⟨étiquette⟩ a disparu de la base de données.
Par ailleurs la suppression a pu entraîner des suppressions d'occurences de relations dont la réalisation est origine ou cible ainsi que des suppressions de réalisations qui ne vérifieraient plus les contraintes des caractéristiques de relations.
Ainsi :
- lorsqu'un ordre suppress a été exécuté, on se trouve dans un contexte auquel n'est plus associé aucune réalisation c'est pourquoi l'ordre suppress s'accompagne obligatoirement d'une clause NEXT ou EXIT qui provoque soit l'itération prématurée soit la sortie du contexte pour le bloc correspondant à la réalisation supprimée.
- l'exécution d'un ordre de suppression peut concerner d'autres réalisations qui peuvent être également supprimées, c'est pourquoi, la règle suivante est imposée :
lorsqu'on supprime une réalisation d'un objet A, le contexte auquel elle est associée ne peut se trouver dans aucun contexte auquel est associée une réalisation suceptible

d'être supprimée avec la suppression de A.

3.4.4.3. Modifications relatives aux objets élémentaires : MODIFY

Forme générale

@ MODIFY [⟨étiquette⟩]$_{o}^{n}$

Cet ordre modifie les valeurs d'O.E. attachées à une réalisation d'O.C. référencée par ⟨etiquette⟩ et conformement aux valeurs associées à cette étiquette dans la zone de communication.

⟨étiquette⟩ est relative à une étiquette active à cet endroit, si cette clause est omise l'ordre se réfère au bloc de base.

3.4.4.4. Création d'occurences de relation : ATTACH.

L'ordre ATTACH permet de créer une occurence de relation déterminée. Une relation doit alors posséder les carctéristiques suivantes : I ⟨ J et K ⟨ L. Si on désire créer plusieurs occurences de relation concernant une même origine , on regroupe ces créations par un ordre ATTACH multiple.

Forme générale :

@ ATTACH [⟨étiquette⟩][,TO ⟨étiq1⟩ VIA ⟨nom relation⟩]$_{1}^{n}$

où ⟨étiquette⟩prend le même sens que précédement.

3.4.4.5. Suppression d'occurences de relation :DETACH.

De même que ATTACH cet ordre ne concerne que des occurences de relations telles que I ⟨ J et K ⟨ L.

Les clauses de l'ordre DETACH spécifient les occurences de relation à supprimer.

Forme générale :

@ DETACH [⟨étiquette⟩] [,FROM ⟨étiq1⟩ ⟨nom de relation⟩]$_{1}^{n}$

3.4.4.6 Modification d'occurences de relation : TRANSFER

Cet ordre permet des modifications d'occurences de relation qui ne peuvent être réalisées par ATTACH et DETACH, c-a-d celles telles que I = J et K = L.

Forme générale :

@ TRANSFER [⟨étiquette⟩] [,FROM ⟨etiq1⟩ TO ⟨étiq2⟩VIA ⟨nom de rel⟩]$_{1}^{n}$

où ⟨étiq1⟩ désigne l'ancienne origine ou cible de ⟨étiquette⟩ et ⟨étiq2⟩ la nouvelle dans la relation nommée.

3.4.5. Traitement des erreurs

Un ordre DML peut se terminer dans trois conditions différentes qui s'expriment par la valeur d'un code de retour de nom SYS-ERROR.
Soit :
- l'exécution s'est exécutée normalement;
- l'exécution ne s'est pas déroulée normalement mais le programme peut se dérouler.
- une erreur grave s'est produite, le programme doit arrêter de travailler sur la base de données.

Les deux premiers cas peuvent être pris en charge par le programme par un text de la valeur de SYS-ERROR.
Le dernier cas provoque automatiquement l'arrêt des communications avec le système SPHINX et son exécution se poursuit à un paragraphe de nom SYS-DUMP qui doit être rédigé par le programmeur.
Exemples :
- la base de données n'existe plus ou est bloquée.
- saturation de la mémoire pour le système SPHINX.
- le programme ne respecte pas la structure de blocs (GO TO et PERFORM illicites).
- ....

3.5. La protection entre programmes concurrents

La gestion de la concurrence des programmes est assurée de façon entièrement automatique par le système SPHINX.
Tout programme DML est rédigé comme s'il devait travailler seul sur les bases de données et le programmeur ne doit pas déclarer d'usage exclusif des données qu'il veut modifier, ni se préoccuper de la résolution des situations d'interblocage.
Cependant la gestion des blocages s'appuie sur une analyse de la structure des programmes.

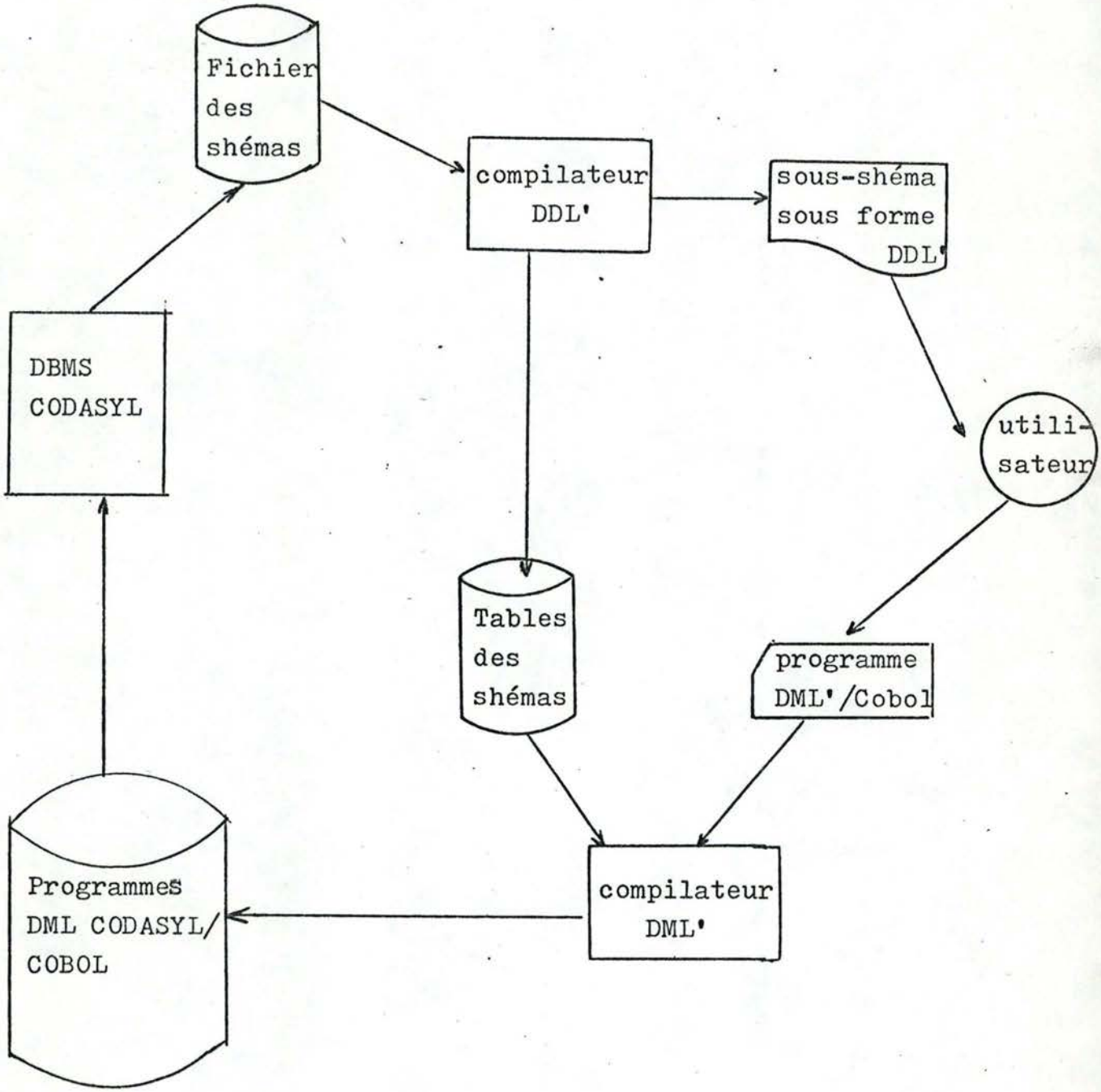
Un programme d'application contenant des ordres DML communique au début d'une section de programme deux listes au système SPHINX :
- une liste d'O.C. aux réalisations desquels le programme va accéder (liste d'accès).
- une liste d'O.C. dont des réalisations peuvent être modifiées (liste de modification).

A la réception de ces listes, ce système suspend l'exécution des programmes jusqu'au moment où d'une part il n'y a plus d'éléments communs entre sa liste d'accès et les listes de modification de tous les autres programmes et d'autre part il n'y a plus d'éléments communs entre sa liste de modification et les listes d'accès de tous les autres programmes.
Dans le DML, les sections de programme correspondent aux blocs REACH-END et PREPARE-END qui ne sont emboîtés que dans des blocs OPEN-CLOSE.

Chapitre 4 : Les langages DDL' et DML'

Ce chapitre décrit brièvement le DDL' ainsi que les différentes étapes d'analyse qui ont abouties à la définition du DML'.
L'idée de base de ce travail est de créer un DDL et DML offrant les avantages des langages SPHINX pour un SGBD répondant aux normes CODASYL. Le travail d'analyse consiste donc à relever dans chacun des systèmes les éléments intéressants (SPHINX) ou indispensables (CODASYL) et de les introduire dans le DML'.
Le texte qui suit présente dans ses lignes générales, le DDL' et d'une manière précise le DML' en justifiant les options prises.

4.1. Fonctionnement général.



Dans un système CODASYL classique, les shémas et sous-shémas sont définis par les administrateurs de la base de données et leurs descriptions sont stockées dans le fichier des shémas.
Lorsqu'un utilisateur désire travailler avec une base de données, il dispose d'une description des sous-shémas qui la composent et celle-ci constitue la base des programmes DML qu'il écrira.
Le DDL' a été conçu en vue d'offrir à un utilisateur une description simplifiée d'un shéma ou sous-shéma CODASYL. Ces simplifications ont pour but de ne faire apparaître que l'aspect logique de la description de la base de données.
Sur base du sous-shéma ainsi décrit, l'utilisateur écrira des programmes DML' de manipulation des données.
Le compilateur DDL' joue un double rôle :

- il décrit les sous-shémas et shémas d'un système CODASYL sous une forme simplifiée (en langage DDL') destinée à un programmeur DML'.
- sur base du fichier des shémas il contruit des tables destinées au compilateur DML' afin de lui faciliter la tâche de contrôle de validité des ordres DML' écrits par un programme utilisateur.

Le compilateur DML' a pour tâche de contrôler la validité des ordres DML' et de générer ensuite l'équivalent CODASYL/COBOL de ces ordres.

1. - Le contrôle de validité d'un ordre DML' est destiné à vérifier sa syntaxe et l'existence des objets manipulés.
2. - Le résultat d'une compilation DML' constitue un programme acceptable par un DBMS CODASYL.

4.2. Le DDL'

La définition de ce langage fait l'objet d'un travail réalisé par M. NGYEN LAN.
Nous en énonçons brièvement les principaux éléments.

4.2.1. Le modèle d'accès décrit par le DDL'

Ce modèle est de même type que celui décrit par le DDL CODASYL : le réseau.
Il décrit des types de record et de set.

Un type de record répond aux mêmes caractéristiques que l'équivalent CODASYL; nous parlerons également de record, data-items, data-aggregates, valeurs de data-item et data-aggregate.
Un type de set représente une relation entre types de record et s'exprime en termes de OWNER et MEMBER.
A un type de set correspond toujours un type de record OWNER et un type de record member. Un set est composé d'un record OWNER et de zéro, un ou plusieurs records members. Lorsque plusieurs types de sets portent le même nom, ils doivents avoir le même type de record OWNER et se distinguent alors par des members différents.
Outre les caractéristiques associées à la notion de set CODASYL, un type de set DDL' présente la particularité suivante :
- un record ne peut être OWNER que d'un seul set parmi ceux dont les types possèdent le même noms et pour lesquels son type de record est déclaré OWNER.

4.2.2. Clauses du DDL'

Clause shéma ou sous-shéma
Elle identifie par un nom unique un shéma ou sous-shéma du système CODASYL.
Clause AREA
Elle identifie par un nom unique une area du sous-shéma.
Clause RECORD
- La description des data-items se fait comme pour les descriptions équivalentes CODASYL.
- Par rapport au DDL CODASYL, n'apparaissent plus les clauses WITHIN, AREA-ID, LOCATION MODE.
- De nouveaux concepts apparaissent et par conséquent de nouvelles clauses.
    - ID-AREA VALUE nom-1, nom-2, ...
      Cette clause définit un data-item fictif (n'existant pas dans la base des données) destiné à contenir le nom d'area relatif à un record.
      Avec la clause value, il indique quelles sont les valeurs possibles pour ce data-item.
      Ce concept regroupe les notions relatives à la clause WITHIN CADASYL.

-IDENTIFIER is data-itl, data-it2, ...
Cette clause déclare que certains data-items sont identifiants pour un type de record.
Elle regroupe les clauses DDL suivantes :
- LOCATION MODE CALC USING data-itl, data-it2, ...
DUPLICATES NOT ALLOWED.
- DUPLICATES NOT ALLOWED FOR data-itl, data-it2, ... pour un record member mandatory d'un set dont l'owner est SYSTEM.
- ACCESS KEY is data-itl, data-it2, ...
Cette clause indique que les valeurs de ces data-items permettent un accès direct pour les records du type concerné.
Elle correspond à la notion de CALC KEY CODASYL.
- RETRIEVAL ONLY.
Cette clause indique que ce record ne peut faire l'objet de modifications. Elle apparaît lorsque certains data-items de contrôle(déclarés comme tels dans le DDL CODASYL) nécessaires à l'exécutions d'ordres de modification ne sont pas déclarés dans le sous-shéma du programme.

Clause SET.
La clause OWNER est identique à celle du DDL CODASYL; la clause MEMBER permet de nommer le type de record member et de le caractériser par des clauses MANDATORY/OPTIONAL et AUTOMATIC/MANUAL comme pour le DDL.
De nouveaux concepts sont introduits par les clauses :
IDENTIFIER is data-itl, data-it2, ...
Cette clause indique que les valeurs des data-items cités sont identifiantes pour les records members d'un set.
Elle regroupe les clauses CODASYL :
- DUPLICATES NOT ALLOWED FOR data-itl, data-it2, ...
- ASCENDING/DESCENDING KEY data-itl, data-it2, ... DUPLICATES NOT.
- SEARCH KEY data-itl, data-it2, ... DUPLICATES NOT.
ACCESS KEY data-itl, data-it2, ...
Cette clause indique que les valeurs des data-items cités permettent l'accès direct à un record member d'un set du type concerné.
Elle correspond à la notion de SEARCH KEY du DDL CODASYL.

On voit donc que les notions de LOCATION MODE DIRECT et VIA et de SET OCCURENCE SELECTION n'apparaissent plus à un utilisateur DDL'.

4.2.3. Conclusions pour le DML'.

Les objets manipulés par les ordres du DML' seront :

- des areas,
- des types de record et de set,
- des data-items,

possédant les caractéristiques décrites par le DDL'.

Le rôle du compilateur DML' sera entre autre de prendre en charge des éléments décrit par le DDL CODASYL et n'apparaissant plus dans le DDL' lorsqu'ils interviennent dans l'écriture d'un ordre DML généré par le compilateur.

4.3. le DML'

Le DML' est un langage de type "structure de boucle" et permet à un utilisateur d'écrire des programmes d'application afin d'obtenir, de modifier, de supprimer ou de créer des données de type CODASYL. Les notions de base du DML' répondant aux mêmes définitions et règles que celles du DML-SPHINX.
Nous utiliserons donc les concepts de structure de bloc, boucle d'accès, bloc ou boucle de base, réalisation courante et étiquette de boucle.

4.3.1. Démarche d'analyse

Les étapes d'analyse qui ont permis la définition du DML sont les suivantes :

- dans un premier temps nous avons comparé les concepts généraux propres aux deux systèmes SPHINX et CODASYL en ne retenant que ceux intervenant au niveau DML. Pour chacune des notions retenues nous avons pris un option en vue de la définition du DML'.
- La deuxième étape d'analyse constitue la définition des ordres du DML' compte tenu des options prises, des objectifs du DML' et des moyens dont dispose le compilateur pour les réaliser.

4.3.2. Première étape

Il apparaît après description des systèmes CODASYL et SPHINX que certains concepts diffèrent ou n'existent pas pour l'un ou l'autre système.
Nous n'analyserons que les notions ayant des répercutions sur l'écriture d'un ordre DML, et pour chacun d'eux la démarche est la suivante :

- les éléments inhérents aux principes de base du DML-SPHINX (stucture de blocs) seront introduits dans le DML' et pris en charge par le compilateur si c'est nécessaire afin de les rendre sous une forme acceptable pour un système CODASYL.
- les élément du système CODASYL seront soit introduits dans le DML' soit pris en charge par le compilateur et par conséquent transparents à l'utilisateur DML'.

Les éléments analysés sont :

- la notion de zone de communication,
- la notion de concurrence entre programme,
- les codes d'erreur à l'exécution d'un ordre DML,
- les notions CODASYL du current et d'area,
- la notion de variable de travail SPHINX.

4.3.2.1. Zone de communication

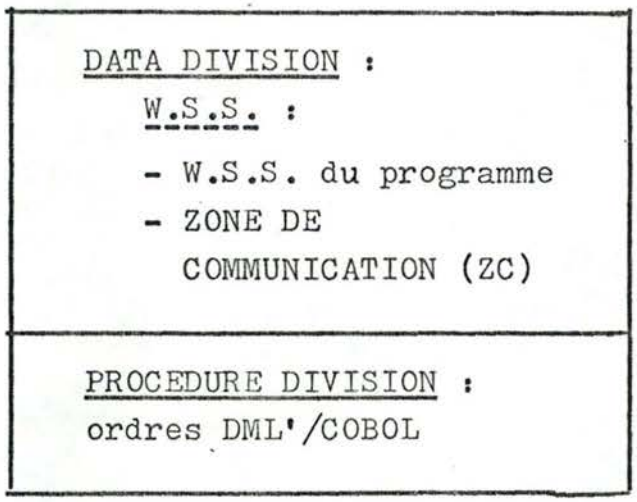
Elle comporte :

pour CODASYL :
- les registres d'état current,
- les registres d'état d'erreur,
- la zone UWA : cette zone est fixe et allouée par le DBMS conformément au sous-shéma du programme : à chaque type de record correspond une place en UWA dont la structure correspond à la description équivalente dans le sous-shéma.

pour SPHINX :
- les variables dans lesquelles le système indique la manière dont l'exécution d'un ordre DML s'est déroulé,
- des paramètres d'exécution des ordres DML à destination du système et auxquelles le programme ne peut avoir accès.
- une zone de données destinée à contenir les valeurs d'O.E. demandées dans les ordres DML d'accès.

Cette zone permet une gestion "dynamique" de la mémoire : chaque ordre définissant une étiquette provoque dans cette zone une réservation de place qui porte le nom de cette étiquette et est destinée à contenir les O.E. dont on demande les valeurs.

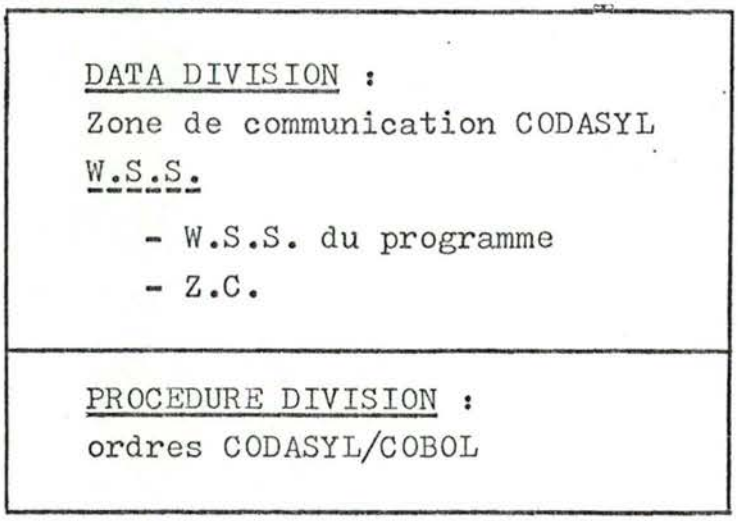
La différence essentielle entre ces deux notions de zone de communication réside dans le fait que SPHINX admet plusieurs réalisations d'un même O.C. simultanément dans cette zone, alors que la zone UWA n'est prévue que pour un seul record d'un type donné.
Le DML' gardera l'avantage offert par SPHINX et un programme DML' aura la structure suivante :



La zone ZC comprendra :

- des variables indiquant la manière dont s'est déroulé un ordre DML' (cfr. codes d'erreur)
- une zone de données destinée à contenir les valeurs des data-items concernés par un ordre DML'. Elle sera établie selon le même principe que la zone de données SPHINX : à chaque étiquette vivante d'un programme correspond une zone de ZC.

Après compilation, le même programme DML' aura la structure suivante :



La zone de communication CODASYL étant inconnue du programmeur DML' le compilateur est chargé de générer des ordres lui permettant de communiquer avec le SGBD CODASYL, c-à-d :

des ordres de transfert de valeurs de ZC vers la zone de communication CODASYL (préalablement à la génération de certains ordres CODASYL ou de la zone de communication CODASYL vers ZC suite à la génération de certains ordres CODASYL.)

4.3.2.2. Concurrence entre programmes

Dans le système SPHINX, le SGBD se charge des problèmes liés à cette notion : les demandes "d'accès" simultanées à une même donnée sont gérées de manière à assurer l'usage exclusif des données utilisées si c'est nécessaire (lors de modifications de données) et à éviter l'interblocage. Un programmeur SPHINX ne doit pas se préoccuper d'assurer la validité des données qu'il manipule.
Le système CODASYL est plus pauvre à ce sujet : un utilisateur est chargé de s'imposer des règles d'utilisation de données et ceci uniquement au niveau global d'une area. Ces règles sont établies au moyen de l'ordre OPEN.
Le langage DML' se voit donc contraint d'opter la solution prise par CODASYL à ce sujet (cfr. chap. 2).

4.3.2.3. Codes d'erreur

Les codes d'erreur à l'exécution d'un ordre DML' seront fonctions des registres d'état d'erreur CODASYL. Nous avons choisi d'introduire deux registres propres au langage DML' : S_ERROR et S_COUNT. S_ERROR rendra compte du dernier type d'erreur détecté dans un ordre DML' et S_COUNT indiquera le nombre total d'erreur détecté pour cet ordre.
Les codes associés à S_ERROR sont :
Code d'avertissement : dans ce cas, un événement a été détecté à l'exécution d'un ordre DML' mais le programme peut continuer en séquence.
Ces événements sont d'une part ceux détectés par le DBMS CODASYL et provoquant un positionnement de ERROR-STATUS = warning et d'autre part ceux détectés par une valeur de ERROR-STATUS = error mais pour des ordres DML' multiples (ex. modify, Detach, Attach...).
Code d'erreur : dans ce cas, un événement a été détecté à l'exécution d'un ordre DML' et celui-ci n'a pu être exécuté ; le programme poursuit son exécution au premier ordre suivant l'ordre de clôture du contexte dans lequel une erreur a été détectée.
Code d'erreur grave : les événements détectés sont tels que tout travail avec la base de données devient impossible au programme. L'exécution se poursuit alors hors du contexte défini par le premier ordre OPEN du programme (≡ sortir du contexte dans lequel le programme travaille avec la base de données).

La création de S_ERROR évite au programme DML' de manipuler des registres CODASYL et d'être confronté à certaines valeurs insignifiantes pour lui, ce sont par exemple les valeurs de ERROR-STATUS relatives à une SOS ou à un location mode CALC ou VIA.

4.3.2.4. Les currents

Un registre current permet à certains ordres CODASYL de référencer directement un record pourvu qu'il contienne sa DBK.

Exemple :

l'ordre FIND OWNER RECORD OF ⟨nom type de set⟩ SET, constitue une demande d'accès au record OWNER d'un set identifié par le current du type de set spécifié.

Le programme devra alors fournir pour ce current la DBK d'un record member du set concerné.

Cette DBK doit donc résulter d'un ordre DML précédent.

Le programme peut en disposer de deux manières :

- elle se trouve encore dans le current concerné, suite à cet ordre préalable;
- elle ne s'y trouve pas ou plus mais elle a été sauvée dans une zone du programme : Z1.

Dans le premier cas, l'ordre FIND peut être exécuté sans mise à jour préalable du current.

Dans le deuxième cas, cette mise à jour s'effectuera par un ordre DML : FIND ⟨nom type record⟩ USING Z1, où ⟨nom type de record⟩ est le nom du type de record member concerné.

Ce dernier ordre FIND a pour effet de vérifier s'il existe un record répondant au critère de sélection et dans ce cas d'effectuer un accès à la base de données et de placer la DBK contenue dans Z1 dans :

- le current de ⟨nom type de record⟩ ,
- tous les currents de type de set pour lesquels le record référencé par Z1 est OWNER ou MEMBER,
- le current de l'area de ce record,
- le current du run-unit.

Le programme peut désirer ne pas modifier la valeur d'un de ces currents (ex. : le current de l'area) car elle contient une DBK qu'il désire garder dans ce current pour l'exécution d'un ordre DML suivant, il écrira alors :

FIND ⟨nom type record⟩ USING Z1 SUPPRESS AREA CURRENCY UPDATES.

On voit par cet exemple, que la notion de current est très importante dans un programme CODASYL car elle constitue la base de nombreux ordres DML et permet l'identification d'un record par un minimum d'informations.

Pour le langage DML', cette notion est très contraignante car elle implique qu'on ne peut travailler simultanément avec plusieurs sets ou records d'un même type et impose donc une gestion très lourde des valeurs de ces currents.

Cette gestion peut être organisée de deux façons :

- une première méthode consiste à mémoriser la liste des currents dont la valeur est susceptible d'être encore utilisée dans la suite du programme, c-à-d : les currents contenant la DBK des records auxquels sont affectées les étiquettes encore vivantes dans un endroit du texte de programme. Lors de la génération d'un ordre DML entrainant une modification de l'un de ces currents, le compilateur accompagne l'ordre généré de la clause SUPPRESS appropriée.
  Ceci n'est pas suffisant dans tous les cas car il peut se présenter qu'un current serve à référencer simultanement deux records. C'est le cas de boucles imbriquées utilisant un même type de set. Dans ces conditions il faut sauver une DBK et procéder comme dans l'exemple pour la faire intervenir par la suite dans un ordre DML.
  La clause SUPPRESS peut se présenter de deux manières :
  - elle cite les currents nécessaires au programme,
  - elle cite tous les currents possibles sauf celui de l'ordre DML concerné.

  Exemple :
  A un endroit du texte du programme, il faut garder les currents des types de set S1,S2,S3 et un ordre DML constitue une demande d'accès à un record OWNER d'un set de type S4; l'ordre peut se présenter sous deux formes :

```
FIND OWNER RECORD OF S4 SET
          SUPPRESS S1, S2, S3 SET
```

  ou

```
FIND OWNER RECORD OF S4 SET
     SUPPRESS ⟨nom de tous les types de set sauf S4⟩
     CURRENCY UPDATES.
```

  Ces deux ordres permettent de conserver les valeurs des currents de S1, S2, S3 et d'obtenir dans le current de S4, la DBK de record OWNER obtenu.
  Remarquons qu'une clause SUPPRESS du deuxième type ne permet pas d'ignorer les currents à conserver dans le programme : un contrôle devra toujours être effectuer afin de vérifier si un current ne sert pas à référencer deux records du programme.

- Une deuxième méthode consiste à se détacher du contexte d'un programme DML et d'ignorer les effets de la mise à jour des currents suite à un ordre DML et par conséquent de ne pas se servir de la clause SUPPRESS.
  Le compilateur génèrera alors :
  + suite à un ordre DML d'accès à un record, un ordre de sauvetage de la DBK de celui-ci dans une zone qui sera directement attachée à l'étiquette correspondante et connue sous la dénomination : D-B-K OF ⟨étiq⟩.
  + préalablement à un ordre DML, un ordre de mise à jour des currents nécessaires.

  Exemple :

  L'ordre DML' : @REACH R1 = ETIQ1 FROM ETIQ2 VIA S1

  où R1 est le nom du type de record OWNER pour le type de set S1 et ETIQ2 fait référence à un record (existant déjà pour le programme) member d'un set de ce type.

  sera généré comme suit :

```
FIND ⟨nom type de record relatif à ETIQ2⟩
USING D-B-K OF ETIQ2.
FIND OWNER RECORD OF S1 SET.
MOVE CURRENCY STATUS FOR RUN-UNIT
TO D-B-K OF ETIQ1.
```

Cette deuxième méthode a été choisie pour le compilateur DML' car elle est plus simple et permet en outre un contrôle à l'exécution : un ordre FIND USING ne s'exécutera qui si le record concerné existe effectivement dans la base de données; le programme est ainsi assuré de n'exécuter des ordres que sur des donnée encore effectives.
On pourrait lui reprocher de générer de nombreux ordres d'accès inutiles, mais un ordre FIND USING ne constituera un accès véritable à la base de données que dans la mesure où le record référencé n'apparaît plus dans les buffers du système.

4.3.2.5. L'AREA

Dans un programme CODASYL cette notion permet à un utilisateur d'accéder à la base de données grâce aux ordres OPEN et CLOSE. Elle permet également de rentabiliser ces accès en n'ouvrant que les areas qui le concernent.
De plus elle constitue le seul élément de la base de données pour lequel il possible d'imposer des régles relatives à la concurrence entre programmes.

A niveau du DML', deux solutions sont envisageables pour la gestion des areas :

solution 1 : elle a pour principe de laisser au compilateur le soin de détecter quels ordres DML' exigeant un ordre OPEN préalable; ceci implique qu'avant chacun de ces ordres il examine les clauses WITHIN concernées et vérifie si les areas concernées ont déjà fait l'objet d'un ordre OPEN. Un ordre CLOSE global serait alors généré en fin de programme.

solution 2 : elle consiste à laisser au programmeur DML' le soin de gérer les areas du programme et donc de mettre à sa disposition des ordres d'ouverture et de fermeture d'areas équivalents aux ordres OPEN et CLOSE CODASYL.

Avantages et Inconvénients

La première solution offre l'avantage à l'utilisateur de le décharger de la gestion des areas mais ses inconvénients sont multiples :

- elle alourdit le travail du compilateur qui sera contraint d'effectuer un test avant chaque ordre DML' et de mémoriser l'état des areas du sous-shéma.
- le compilateur ne travaille qu'au niveau de la clause WITHIN et pour un type de record concerné par un programme DML', il générera un ordre OPEN pour toutes les areas de cette clause; ceci retire au programmeur DML' l'avantage de ne travailler qu'avec les areas qui le concerne.
- la notion d'area constitue pour l'utilisateur le seul moyen d'imposer des régles d'usage des données afin de régler les problèmes de concurrence vis à vis d'autres utilisateurs. Lorsque le compilateur générera un ordre OPEN il sera contraint de prendre à ce

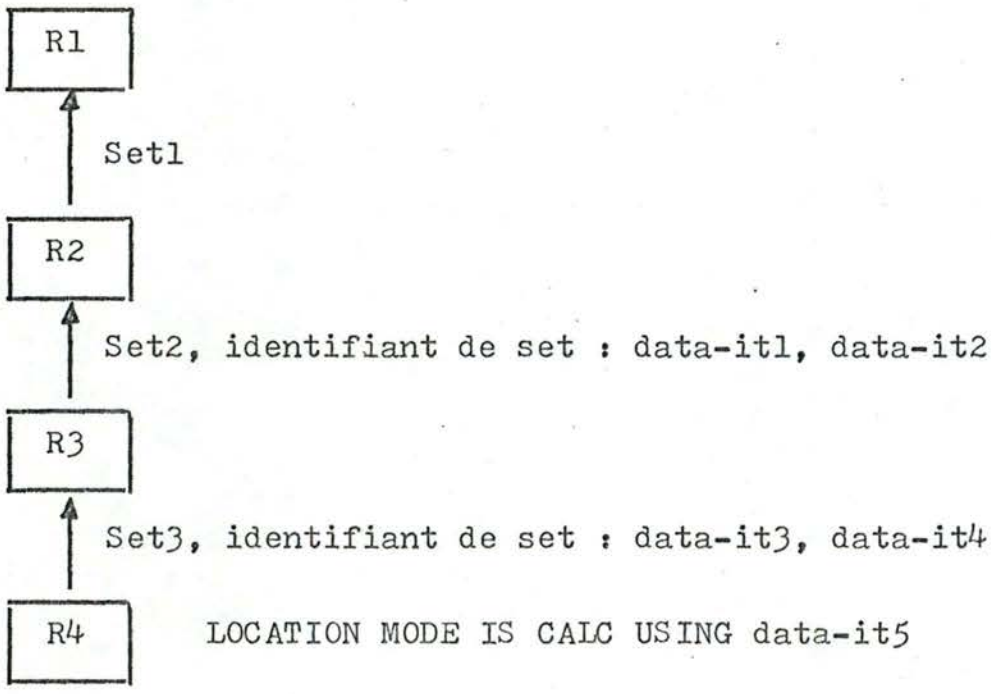
sujet des options qui ne correspondront pas nécessairement aux intentions de l'utilisateur DML'.

- Cette solution n'évite pas au progammeur DML' d'ignorer la notion d'area : lors de la création d'un record il est seul à pouvoir déterminer l'area dans laquelle doit se placer ce record.

La deuxième solution apparaît plus intéressante mais elle pose toutefois un problème lors de génération d'ordres impliquant une SOS : cette notion étant inconnue du programmeur DML', le compilateur sera chargé de générer des ordres de reconstitution du chemin d'accès décrit par une SOS.

Exemple :

Lors de la création d'un record R1, il faut le rattacher à un record R2 OWNER d'un set pour lequel R1 est mandatory automatic member.
La SOS associée à ce set est la suivante :



La SOS implique que pour identifier le set1, il faut :

- identifier R2 en tantque member de set2 par ses valeurs pour data-it1 et data-it2.
- identifier l'OWNER R3de set2 en tant que member de set3 par ses valeur pour data-it3 et data-it4.
- identifier l'OWNER R4 de set3 par sa valeur de CALC KEY : data-it5.

Un programmeur DML' ne fournit que les records R1 et R2; le compilateur doit donc, pour reconstituer cette SOS, générer les ordres suivants :

- FIND ⟨type de record de R2⟩ RECORD USING D-B-K OF ⟨étiquette relative à R2⟩
- placer les valeurs de data-it1 et data-it2 en UWA
- FIND OWNER RECORD OF ⟨type de set de SET2⟩ SET
- placer les valeurs de data-it3 et data-it4 en UWA
- FIND OWNER RECORD OF ⟨type de set de SET3⟩ SET
- placer la valeur de data-it5 en UWA
- si il existe une clause AREA-ID pour ce type de record : placer la valeur de AREA-NAME dans la zone définie par cette clause .

Chacun des ordres FIND OWNER impliquent que l'area du record OWNER ait préalablement fait l'objet d'un ordre OPEN et seul le compilateur peut déterminer quelles sont ces areas.

Choix d'une solution

Le choix d'une des deux solutions peut faire l'objet d'une étude approfondie qui attribuerait un facteur de pondération à chaque avantages et inconvénients cités.
Cette pondération tiendrait compte des désirs des utilisateurs, de la probabilité de retrouver des clauses WITHIN avec plusieurs nom d'areas.
Une telle étude sort du cadre de ce mémoire et la solution choisie ne tient compte que de considérations très générales :

- l'utilisateur est plus à même de déterminer quelles sont les areas concernées par un programme DML' ainsi que de placer aux endroits requis des ordres d'ouverture et de fermeture.
- le programmeur DML' ne doit pas être désavantagé par rapport à tout autre utilisateur CODASYL.

La deuxième solution a donc été adoptée et le langage DML' comportera des ordres OPEN et CLOSE équivalents aux ordres CODASYL.
Lorsqu'un ordre DML' implique une SOS, le compilateur produira un message destiné à l'utilisateur lui indiquant quelles sont les areas éventuelles à ouvrir.

4.3.2.6. Les variables de travail

Un programmeur SPHINX dispose de 99 variables de travail définies par le système. Elles sont destinées à contenir la référence à une réalisation d'O.C. ainsi que le type qui lui correspond.
Les ordres DML de manipulation de ses variables sont :

- SAVE : affectation d'une valeur à une telle variable.
- TEST : test si la variable est initialisée ou non.
- MOVE : transfert de la valeur d'une variable de travail dans une autre.
- CLEAR : vider une variable de son contenu.

Dans le système CODASYL une telle notion se rapproche du concept de DBK.
Pour le DML' une variable de travail devra donc contenir une DBK.
Le langage COBOL étendu au DML CODASYL permet de définir une variable de type DBK, c'est pourquoi le programmeur DML' définira ses propres variables de travail. Ceci lui offre la possibilité de les définir en nombre et sous la forme voulue, il pourra par exemple créer des tableaux de variables de travail, insérer une variable de travail dans un groupe, ...
Les ordres DML SPHINX de manipulation de ces variables deviendront inutiles car le langage COBOL permet la manipulation de telles variables sans les distinguer des autres.
On gardera toutefois l'ordre d'affectation (SAVE) de valeur à une variable de travail car un programme identifie un record par le nom d'étiquette correspondant et l'ordre SAVE prend la forme suivante :

@ SAVE <étiquette> IN <variable de travail>

Le compilateur générera pour cet ordre :

MOVE D-B-K OF <étiquette> TO
<variable de travail>

Il existera également l'ordre REACH de format 2 qui permet l'accès à un record à partir d'une variable de travail :

@ REACH <nom type record> <var. travail> = <étiquette>

4.3.3. Deuxième étape

Cette deuxième étape consiste à analyser chaque ordre du langage. La définition d'un ordre DML' tient compte des options prises à la prmière étape d'analyse et des ordres CODASYL/COBOL qui seront générés.

La présentation d'un ordre comportera trois parties :

1.)- Ordre DML' : ce paragraphe définit l'ordre dans sa syntaxe et sémantique et décrit les valeurs de S-ERROR qui lui sont associées.

2.)- Ordre CODASYL : ce paragraphe présente l'ordre CODASYL équivalent (s'il en existe un) qui sera la base du texte généré. Dans la mesure où elles présentent des différences avec l'ordre DML' ou permettent de définir le travail du compilateur, nous présentons les régles et les valeurs de ERROR-STATUS associées à cet ordre.

3.)- Justification de l'ordre DML' : ce paragraphe justifie certaines options prises pour la définition et/ou la génération de l'ordre DML'.

Le texte qui suit présente chacun des ordres après avoir défini quelques conventions d'écriture.

4.3.3.1. Régles d'interférence DML'/COBOL

Les principe du DML' proposent au programmeur le respect de certaines régles de construction des programmes :

- il est déconseillé d'entrer dans un contexte défini par des ordres d'accès (REACH-OPEN-PREPARE) par un ordre autre que ceux-ci.
  Aucun contrôle n'est effectué à la compilation et à l'exécution. Un utilisateur ne respectant pas cette règle prend le risque de travailler avec des valeurs insignifiantes;
- il est également déconseillé de ne sortir d'un contexte que par les ordres END et CLOSE; un programme ne respectant pas cette règle n'exécute pas les opérations associées à ces ordres.

4.3.3.2. Conventions utilisées pour définir la syntaxe

Les Symboles

@ = < > ,; . : ( )

Mots

Verbes : OPEN, CLOSE, REACH, NEXT, EXIT, END, PREPARE, CREATE, MODIFY, ATTACH, DETACH, TRANSFER, SAVE.

Opérateur logique : AND

Opérateur de négation : NOT

Mots fonctionnels : FROM, GET, TO, VIA, IF, NEXT, EXIT, ALL, ONLY, SELECTIVE, USAGE MODE, RETRIEVAL, UPDATE, EXCLUSIVE.

Identificateurs :

- nom de zones COBOL
- noms d'éléments du modèle (types de records et de sets, data-items et areas, shémas et sous-shémas).
- noms d'étiquettes DML' : obéissent à la règle de formation des identificateurs COBOL, mais on une longueur de 8 caractères au plus. Une étiquette est unique dans un programme.
- noms de variables de travail : obéissent à la règle de formation des identificateurs COBOL.

Constantes numériques et alphanumériques : mêmes formats que ceux accepté par le compilateur COBOL, chaines de caractères limités à 56 positions.

Règle de séparation des mots

Le compilateur DML' est capable de distinguer parmi deux éléments consécutifs d'une même ligne :

- un symbole d'un mot, même s'ils sont contigus;
- deux symboles même s'ils sont contigus;
- deux mots, s'ils sont séparés par au moins un espace.

Les Métasymboles

L'expression de la syntaxe DML' fait appel à plusieurs métasymboles dont le rôle est de désigner soit des classes d'identificateurs, soit des clauses qui devront être développées par la suite; ils apparaîtront entre < et > .

Classes d'identificateurs

Noms d'éléments du modèle : <nom shéma>
<nom sous-shéma>
<nom type de record>
<nom type de set>
<data-item>

Noms d'étiquettes : <étiq> , <étiq1> , <étiq2> .

Noms de variables de travail : <var. trav.> .

Clauses à développer

<des. valeur> : mis pour : soit une constante
soit le nom d'une zone COBOL.

<liste des.valeur> : liste d'une ou plusieurs <dés. valeur> .

<condition> : mis pour une condition intervenant dans un ordre REACH.

Les expressions syntaxiques

Tout ordre DML' se décompose en un certain nombre de clauses. Une clause peut elle même se décomposer en d'autres clauses plus simples.

Une clause est une suite significative de symboles permettant de définir un verbe, un identificateur, un mot fonctionnel, un méta-symbole ... .

Le texte d'une expression syntaxique se lit de la gauche vers la droite et certaines clauses peuvent être regroupées par les symboles : [ ] , { } , ‖ ‖ dont les significations sont alors :

1.- [a b c] Aucune de ces trois clauses (a, b, c) ne sont obligatoires et l'ordre DML' peut comporter au plus une des trois clauses.

2.- {a b c} Une et une seule de ces trois clauses doit être présente dans l'ordre.

3.- ‖a b c‖ L'ordre DML' doit comporter au moins une de ces clauses et au plus une occurence de chacune d'elles.

Un mot souligné dans un ordre signifie que le mot est réservé au langage et est obligatoire dans la clause dans laquelle il apparaît.

4.3.3.3. Les ordres du DML'.

Chacun de ordres est décrit selon les trois aspects décrits précédemment.
Les valeurs de S-ERROR, indiquant un code d'erreur grave, sont décrites dans le paragraphe 4.3.3.4.; elles sont valables pour tout ordre DML'.

OPEN

Cet ordre constitue l'ordre d'accès à la base de données.
Dans un système CODASYL cet accès s'effectue par deux ordres :

- lordre INVOKE, destiné à la compilation, référence un sous-shéma et a pour effet de réserver de la place mémoire pour les données ainsi décrites.
- les areas de ce sous-shéma sont accessibles par un ordre OPEN.

Dans une première étape d'analyse nous avons décidé de laisser au programme DML' le soin de gérer les areas qu'il manipule; c'est pourquoi il doit exister des ordres DML' OPEN et CLOSE d'areas.
Compte tenu de la structure de bloc, il est nécessaire de déterminer un contexte pour le programme à l'intérieur duquel il est possible d'accéder à la base de données.
Les ordres OPEN et CLOSE DML' se présentent donc sous deux formes et remplissent d'une par la fonction d'accès aux areas et d'autre part celle de référencer un sous-shéma et de déterminer le contexte qui lui est associé.

1. Forme 1

1.1. Ordre DML'

Syntaxe :

@ OPEN SUB-SHEMA ⟨nom de sous-shéma⟩ = ⟨Etiq⟩ OF SHEMA ⟨nom de shéma⟩ .

Règles syntaxiques

1.- Le sous-shéma et shéma cités doivent être connus du DBMS.
2.- Il ne peut exister qu'un seul ordre de ce type dans un programme.
3.- Cet ordre décrit, avec l'ordre OPEN correspondant, un contexte hors duquel aucun ordre DML' ne peut être écrit.

Sémantique

1.- L'ordre OPEN de forme 1 est l'ordre d'en-tête de la boucle déterminant le contexte du programme travaillant avec le sous-shéma cité dans cet ordre.
2.- Il détermine les types des données manipulées par le programme.

1.2. Ordre CODASYL

L'ordre INVOKE permet à un programme de nommer le sous-shéma auquel il se réfère.

Forme générale :

INVOKE SUB-SHEMA ⟨nom sous-shéma⟩ OF SHEMA ⟨nom shéma⟩ .

2. F o r m e 2

Cette forme de l'ordre OPEN permet l'ouverture des areas du sous-shéma d'un programme. Cet ordre ne diffère pas de l'ordre OPEN CODASYL dans ses règles syntaxiques et sémantiques, c'est pourquoi nous présentons uniquement l'ordre DML'.

Syntaxe :

@ OPEN { ALL [FOR SET ⟨nom type de set1⟩ , ⟨nom type de set2⟩] / AREA ⟨nom area1⟩ , ⟨nom area2⟩ ... }

[USAGE MODE IS [EXCLUSIVE / PROTECTED] { RETRIEVAL / UPDATE }]

Règles syntaxiques

1.- Les types de sets spécifiés et les areas citées doivent appartenir au sous-shéma du programme.

2.- A un ordre OPEN peuvent correspondre plusieurs ordres CLOSE de forme 2.

Règles sémantiques

1.- Cet ordre provoque l'ouverture de certaines areas spécifiées par cet ordre et définit l'usage mode associé à celles-ci.

2.- Si cet ordre s'accompagne de la clause :

ALL FOR SET : - les areas concernées sont celles contenant les records appartenant aux types de set spécifiés;
- sans la clause FOR SET, les areas concernées sont toutes celles du sous-shéma.

AREA : les areas concernées sont celles explicitement citées dans l'ordre.

RETRIEVAL : le programme permet à tout autre programme d'ouvrir ces areas avec un usage mode autre que exclusive.

UPDATE : le programme permet à tout autre programme d'ouvrir ces areas avec un usage mode autre que exclusive ou protected.

EXCLUSIVE : cette clause empêche à tout autre programme l'accès à ces areas.

PROTECTED : cette clause permet à tout autre programme l'accès à ces areas uniquement avec un usage mode RETRIEVAL (sans exclusive ou protected).

3.- Par défaut, la clause usage mode est RETRIEVAL.

4.- L'ordre OPEN, ainsi que l'USAGE MODE associé, reste en effet jusqu'à l'ordre CLOSE relatif à ces areas.

5.- Quelles sont les areas qui doivent être ouvertes ?
Les areas contenant les record faisant l'objet d'ordres REACH, CREATE et CLOSE ainsi que les records faisant implicitement l'objet d'un ordre suppress ou detach suite à un ordre suppress ou detach écrit dans le programme.
Les areas contenant des records constituant un chemin d'accès à un set pour un ordre CREATE ou TRANSFER impliquant une sélection d'un set pour un record mandatory member.

6.- Pour l'exécution d'un ordre ATTACH, DETACH, CREATE, SUPPRESS, MODIFY, TRANSFER, les areas concernées doivent être ouvertes avec un usage mode UPDATE.

7.- Lors de l'ouverture d'areas, il peut se faire que l'usage mode associé à cet ordre entre en conflit avec un usage mode associée aux mêmes areas ouvertes par un autre programme.
Les cas de conflit (c) sont les suivants :

	1	2	3	4	5	6
1. RETRIEVAL					c	c
2. UPDATE			c	c	c	c
3. PROTECTED UPDATE		c	c	c	c	c
4. PROTECTED RETRIEVAL		c	c		c	c
5. EXCLUSIVE RETRIEVAL	c	c	c	c	c	c
6. EXCLUSIVE UPDATE	c	c	c	c	c	c

L'action prise en cas de conflit est propre à chaque implémentation du DBMS.

Valeurs de S-ERROR

Code d'erreur

0001 : une ou plusieurs areas concernées dans l'ordre sont déjà ouvertes.

0002 : le programme n'a pas satisfait les locks nécessaires pour cet ordre.

0003 : l'utilisation de la clause EXCLUSIVE ou PROTECTED peut amener un état d'interblocage.

0004 : l'area est inconnue.

0005 : l'area est indisponible (OFF-line).

CLOSE

Cet ordre est associé à l'ordre OPEN c'est pourquoi il se présente également sous deux formes.

1. Forme 1

Associée à un OPEN de forme 1, il clôture un contexte constitué des ordres compris entre cet ordre et l'OPEN correspondant.
Il n'existe pas d'ordre CODASYL équivalent.

Syntaxe

@ CLOSE [<étiq>].

Règles

1. - <étiq> ne remplit aucune fonction particulière et n'est pas obligatoire.
2. - <étiq> doit exister pour le programme et est l'étiquette définie par l'ordre OPEN de formel du programme.
3. - Il ne peut exister qu'un seul ordre CLOSE de cette forme dans un programme.

Règles sémantiques

1. - Ce CLOSE définit la fin du contexte ouvert par l'ordre OPEN de formel lui correspondant.
2. - Après son exécution, le sous-shéma devient indisponible au programme.
3. - Il clôture également tous les contextes qui lui sont emboités et qui n'auraient pas encore été fermés à cet endroit.
   Ces contextes ont été ouverts par des ordres REACH ou PREPARE.
   Pour ces contextes, l'ordre CLOSE définit une suite de clôture END emplicites.
4. - Il provoque en outre la fermeture de toutes les areas encore ouvertes à cet endroit.

2. Forme 2

Sous cette forme, l'ordre CLOSE commande la fermeture d'areas ouvertes par des ordres OPEN de forme 2. Il possède les règles syntaxiques et sémantiques du même ordre CODASYL, c'est pourquoi nous présentons directement l'ordre DML'.

Syntaxe

@ CLOSE { ALL [FOR SET <nom type de set1> , <nom type de set2> ...]
        { AREA <nom area1> , <nom area2> ... .

Règle syntaxique

Toutes les areas ainsi que les types de set cités doivent être déclarés dans le sous-shéma du programme.

Règles sémantiques

1. - Après l'ordre CLOSE, toute tentative d'accès aux areas concernées devient impossible et l'effet de l'usage mode s'annule.
2. - Si cet ordre s'accompagne de la clause :
   FOR SET : les areas concernées sont toutes celles contenant des records participants aux types de sets spécifiés.
   ALL : les areas concernées sont toutes celles du sous-shéma.
   AREA : les areas concernées sont celles explicitement citées.

Valeurs de S-ERROR

Code d'erreur

0101 : une ou plusieurs areas n'étaient pas ouvertes.

3. Justification des ordres OPEN-CLOSE DML'

3.1. Forme 1

L'étiquette crée par l'ordre OPEN peut paraître inutile étant donné qu'un programme ne travaille qu'avec un seul sous-shéma.
Nous avons gardé l'étiquette de cet ordre par soucis d'uniformité des ordres d'accès DML' et l'étiquette pourra ainsi intervenir dans un ordre EXIT.

3.2. Forme 2

Nous avons choisi de ne pas générer une boucle d'accès à partir des ordres OPEN et CLOSE de forme 2 pour deux raisons :
- Etant donné les règles de concurrence associées à une area, il est nécessaire de permettre à un utilisateur de libérer le plus rapidement possbile les areas dont il ne se sert plus.

- les areas nécessaires à un programme n'apparaissent pas toujours explicitement à un utilisateur, c'est le cas d'ordres DML' impliquant une SOS.
  Le contexte d'utilisation d'une area ne peut donc être aussi clairement défini que celui d'un sous-shéma ou d'un type de record.

3.3. Les codes d'erreurs indiquent que l'ordre n'est pas exécuté mais le programme peut continuer en séquence. Ceci se justifie par le fait que les ordres OPEN et CLOSE de forme 2 ne constituent pas une boucle d'accès.

REACH

1. Ordre DML'

Syntaxe

Format 1

@ REACH ⟨nom type de record⟩ = ⟨Etiq1⟩
[[FROM ⟨Etiq2⟩] VIA ⟨nom type de set⟩]
[IF (⟨condition⟩)] [GET].

Format 2

@ REACH ⟨nom type de record⟩ ⟨var. travail⟩ = ⟨Etiq1⟩
[IF (⟨condition⟩)] [GET]

Règles syntaxiques

- ⟨nom type de record⟩ : est le nom d'un type de record déclaré dans le sous-shéma du programme.
- ⟨nom type de set⟩ : est le nom d'un type de set déclaré dans le sous-shéma du programme.
- ⟨étiq1⟩ : est le nom de l'étiquette de boucle qui sera attachée au contexte défini par cet ordre; ce nom doit être unique dans le programme.
- ⟨étiq2⟩ : est le nom d'une étiquette de boucle attachée à un contexte dans lequel se trouve cet ordre et si la clause FROM est omise l'ordre désigne implicitement son contexte de base. Le contexte ainsi défini est relatif à un ordre REACH ou PREPARE suivi d'un ordre CREATE.
- ⟨condition⟩ : est la description d'une expression booléenne de critères simples.

Les expressions booléennes sont limitées aux formes suivantes :
- critère simple
- produit logique de critères simples (critères unis par des AND).

- Un critère simple est constitué de la manière suivante :

soit ⟨nom data-item⟩ ⟨op1⟩ ⟨dés. valeur⟩
soit ⟨nom data-item⟩ ⟨op2⟩ ⟨liste dés. valeurs⟩

où :

- ⟨nom data-item⟩ : est le nom d'un data-item déclaré dans le sous-shéma du programme appartenant au type de record désigné.
- ⟨op1⟩ : est l'un des opérateurs de comparaison : =, NOT =, < , NOT < , > , NOT > ,
- ⟨op2⟩ : est l'un des opérateurs de comparaison : =, NOT = .
- ⟨dés. valeur⟩ : est la désignation d'une valeur, soit sous la forme constante (littéral numérique ou alphanumérique COBOL), de nom de zone COBOL ou de nom de zone dans la zone de communication.
- ⟨liste des valeurs⟩ : est une liste de ⟨dés. valeurs⟩ séparées par des virgules.
- ⟨var. travail⟩ : est le nom d'une variable de travail.
- La clause VIA est obligatoire s'il existe une clause FROM.
- Le type de record spécifié doit être OWNER ou MEMBER du type de Set désigné :
  - s'il est déclaré MEMBER : la clause ⟨étiq2⟩ (explicite ou implicite) doit référencer le record OWNER, sauf si celui-ci est déclaré SYSTEM, dans ce cas la clause FROM doit être omise et est considérée comme inexistante;
  - s'il est déclaré OWNER : la clause ⟨étiq2⟩ (explicite ou implicite) doit référencer un record MEMBER.
- La clause GET est implicite s'il existe une clause IF.

Sémantique

Format 1

Cette forme de l'ordre REACH commande l'accès à des records du type désigné.
Parmi ces records, seuls seront retenus ceux auxquels sont associées des valeurs de data-items obéissant aux clauses IF et VIA éventuelles.
Un record ainsi sélectionné constituera une réalisation courante de la boucle qui sera affectée à ⟨étiq1⟩ .
Le corps de la boucle sera exécuté pour chaque réalisation courante, c-à-d : zéro, une ou plusieurs fois selon les cas.
L'accès aux records sera réalisé différemment selon les clauses présentes dans l'ordre.
On distingue deux modes d'accès :

soit : l'accès est réalisé à partir d'un set = ordre REACH avec clause VIA et FROM (implicite ou explicite);
alors : - si ⟨nom type de record⟩ est un type de record MEMBER pour le type de set ⟨nom type de set⟩ , on accèdera séquentiellement à tous les records de ce type qui sont members effectifs d'un set de ce type, celui-ci étant déterminé par la clause ⟨étiq2⟩ référençant implicitement ou explicitement le record OWNER; ceci ne s'applique pas dans le cas ou l'OWNER est désigné SYSTEM car alors la clause ⟨étiq2⟩ ne peut exister : l'occurence de ce set est unique.
- si ⟨nom Record⟩ est un type de record OWNER pour le type de set désigné , on accèdera au record OWNER d'un set de ce type, celui-ci étant déterminé implicitement ou explicitement par la clause FROM qui référence un record MEMBER.

soit : l'accès sera réalisé pour les records du type désigné. Cet accès est alors basé sur l'appartenance de ces records à une (ou plusieurs) area(s) = ordre REACH sans clause VIA.
Des tentatives d'accès seront effectuées pour toutes les areas susceptibles de contenir ce type de record et seules seront examinées les areas qui auront fait l'objet d'un ordre OPEN préalablement dans le programme.

Format 2

Cette forme de l'ordre REACH commande l'accès à un record référencé par ⟨var. travail⟩ pourvu que cette dernière ait été initialisée et que son contenu spécifie un record du type désigné.
Si un tel record peut être obtenu, il constituera la réalisation courante de la boucle et sera affecté à ⟨étiql⟩ et le corp de la boucle sera alors exécuté une fois, sinon la boucle ne sera pas exécutée.

Clause IF

L'interprétation de la condition correspond à celle qui est d'usage en logique et en particulier dans la condition COBOL.
Un critère faisant intervenir une ⟨liste dés. valeur⟩ s'interprète comme suit :

- ⟨nom data-item⟩ = ⟨liste dés. valeurs⟩ : la valeur du data-item ⟨nom data-item⟩ attachée à ⟨étiql⟩ appartient à la liste des valeurs désignées.
- ⟨nom data-item⟩ not = ⟨liste dés. valeurs⟩ : la valeur du data-item ⟨nom data-item⟩ attachée à ⟨étiql⟩ n'appartient pas à la liste des valeurs désignées.
- si ⟨nom data item⟩ = ID-Area :
  - il ne peut apparaître que dans un seul critère simple de la condition.
  - l'opérateur de ce critère ne peut être que : = ou not = .

Clause GET

Cette clause donne accès à toutes les valeurs des data items attachées à la réalisation courante de boucle affectée à ⟨étiql⟩ .

Contexte

L'ordre REACH définit un contexte qui s'étend jusqu'au END qui lui correspond en aval du programme.

COUNT

A chaque boucle définie par un ordre REACH est associé un compteur de réalisations référençable par le programme sous le nom de COUNT-⟨étiql⟩

Ce compteur rend compte du nombre de réalisations courantes de la dernière exécution de la boucle.
Il est accessible à tout endroit du programme inclus dans le contexte de la boucle d'accès OPEN-CLOSE du programme.

Valeurs de S-ERROR

Format 1

Codes d'erreur

0201 : pour un ordre REACH avec clause VIA, la dernière réalisation de boucle n'existe plus ou n'est plus accessible car son area est fermée.
0202 : le record référencé par ⟨étiq2⟩ est inconnu ou n'existe plus.
0203 : le record référencé par ⟨étiq2⟩ n'est pas member d'un set du type spécifié.
0204 : ou se trouve dans une area qui n'est pas ouverte.
0205 : ce programme n'a pas satisfait aux locks.
0206 : pour un ordre REACH avec clause VIA : l'OWNER, le premier record ou le member suivant la dernière réalisation de boucle dans le set se trouve dans une area qui n'est pas ouverte.
0207 : ou sous contrôle exclusif ou OFF-line.

Codes d'avertissement

0208 : la dernière réalisation de boucle ou le record référencé par ⟨étiq2⟩ n'est plus member d'un set de type spécifié mais l'accès est effectué.
0209 : pour un ordre REACH sans clause VIA, les records du type spécifié et répondant aux critères de sélection ne peuvent tous être obtenus car les areas nécessaires ne sont pas toutes ouvertes.
0210 : ou sous contrôle exclusif ou OFF-line.
0211 : ou lors de l'accès à une area particulière, la dernière réalisation de boucle appartenant à cette area n'existe plus et l'accès à cette area est alors abandonné.

Format 2

Codes d'erreur

0212 : le record référencé par la variable de travail est inconnu, n'existe pas ou n'est pas du type spécifié.

0213 : ou se trouve dans une area qui n'est pas ouverte.

Valeurs relatives à la clause GET

Code d'avertissement

0301 : le DBMS ne peut faire la conversion entre un data-item défini par le sous-shéma et sa forme dans le shéma; la clause GET est alors sans effet.

0302 : le programme n'a pas fourni les LOCKS nécessaires.

0303 : la valeur d'un data-item déclaré VIRTUAL SOURCE/RESULT ne peut être obtenue et sera remplacé par une valeur nulle dans la zone de communication.

0304 : après application de la clause CHECK, une valeur de data-item déclaré VIRTUAL/RESULT est considéré invalide mais elle est placée dans la zone de communication.

0305 : la conversion entre shéma et sous-shéma provoque la troncation de valeurs significatives.

2. Ordre CODASYL

Il n'existe pas d'ordres équivalent en CODASYL et le programme DML doit produire une séquence d'ordres d'accès de test de retour et de branchement d'itération pour obtenir un texte équivalent.
Les ordres d'accès CODASYL sont :

2.1. Accès basé sur une valeur de DBK

- FIND <nom de type de record> USING Z1
  et
  FIND CURRENT OF { <nom de type de record> RECORD. / <nom de type de set> SET. / <nom d'area> AREA. / RUN-UNIT. }
  où la DBK du record concerné est soit à fournir par le programme grâce à la zone du programme : Z1, soit contenue dans un current désigné dans l'ordre.
- Valeurs de ERROR-STATUS :

+ E R R O R : 0301 : l'area concernée n'est pas ouverte
0318 : ou OFF-line ou sous contrôle exclusif d'un autre programme.
0310 : le programme n'a pas fourni les LOCKS nécessaires à cet ordre.
0326 : le record référencé par Z1 n'est pas du type spécifié dans l'ordre.
0306 : la DBK fournie est inconnue ou nulle.
0317 : si l'ordre référence explicitement un current et que le record ainsi concerné n'existe plus dans la base de données.

+ W A R N I N G : 0350 : si un current de type de set est spécifié et le record ainsi référencé n'appartient plus effectivement à ce type de set.

2.2. Accès aux records d'un set

2.2.1. Accès à l'OWNER

- FIND OWNER RECORD OF ⟨nom de type de set⟩
  ou
  FIND OWNER IN ⟨nom de type de set⟩ CURRENT OF

⟨nom type de record⟩	RECORD
⟨nom type de set⟩	SET
⟨nom d'area⟩	AREA
RUN-UNIT	

  Le current du type de set concerné contiendra la DBK d'un record member pour le premier ordre.
  Pour le deuxième ordre le current explicitement concerné contiendra la DBK d'un record member.
- Valeurs de ERROR-STATUS :

+ E R R O R : 0301,0318,0310,0306,0317 : comme précédemment.
0322 : le record référencé par current n'est pas member effectif du set concerné.

+ W A R N I N G : 0350 : comme précédemment.
0351 : le record référencé implicitement par current n'existe plus dans la base de données.

2.2.2. Accès à un record MEMBER

- FIND { NEXT | PRIOR | FIRST | LAST | n | var } ⟨nom de type de record⟩ RECORD OF ⟨nom type de set⟩ SET.

  Le current du type de set concerné contiendra la DBK d'un record member ou owner, identifiant ainsi le set de ce type sur lequel porte l'ordre si on a la clause :
  NEXT (ou PRIOR) : on accède au record member suivant (ou précédent) le record, spécifié par current, dans le set. Si ce dernier est OWNER pour le set, on accède alors au premier (ou dernier) record du set.
  FIRST (ou LAST) : on accède au premier (ou dernier) record member du set concerné.

n (ou VAR) : n (ou la valeur de VAR) est un littéral numérique : i et on accède alors au $i^{ème}$ record member pour le set concerné. Si la clause ⟨nom de type de record⟩ est présente, on n'accède qu'aux records members du type spécifié.

- Valeurs de ERROR-STATUS :

+ E R R O R : 0301,0318,0310,0306,0322 : comme précédemment.
0326 : le set concerné est vide.
0307 : le record spécifié par current est le dernier du set (pour NEXT) ou le premier du set (pour PRIOR) ou si i indique une valeur supérieure au nombre de records members du set concerné (pour n ou VAR).

+ W A R N I N G : 0350, 0351 : comme précédemment.

- FIND ⟨nom de type de record⟩ VIA [CURRENT OF] ⟨nom type de set⟩ USING ⟨nom de data-item1⟩ , ⟨nom de data-item2⟩ , ...
Cet ordre donne accès au premier record member d'un set de type défini, pour lequel les valeurs de data-item1, data-item2, ... sont égales aux valeurs présentes dans les zones correspondantes en UWA.
Le set concerné est identifié par le current du type de set (avec clause CURRENT OF) soit par la SOS associée à ce set (sans clause CURRENT OF); dans le premier cas le programme fournira la DBK d'un record OWNER ou MEMBER du set et dansle deuxième cas il fournira toutes les valeurs (DBK, identifiants de set) nécessaires à la reconstitution de cet SOS.

- Valeurs de ERROR-STATUS : 0301, 0318, 0310,0306 : comme précédemment

+ E R R O R : 0323 : si la SOS implique une clause AREA-ID et que le programme spécifie un nom d'area inexistant pour la clause WITHIN concernée.
0302 : si la SOS implique une clause LOCATION MODE CALC avec AREA-ID et qu'aucun record ne peut être obtenu avec les valeurs fournies pour ces deux clauses.

0326 : aucun record member de set spécifié ne possède de telles valeurs pour les data-items spécifiés.

+ W A R N I N G : 0350, 0351 : comme précédemment.

- FIND NEXT DUPLICATES WITHIN ⟨nom de type de set⟩ USING ⟨nom de data-item1⟩ , ⟨nom de data-item2⟩ , ...
  Le current du type de set spécifié contiendra la DBK d'un record member et identifiera ainsi le set concerné.
  Cet ordre donne accès au premier record, member du set concerné, à partir du record spécifié par current et qui est du même type et possède les mêmes valeurs pour data-item1, data-item2 que le record référencé par current .
- Valeurs de ERROR-STATUS

+ E R R O R : 0301, 0318, 0310, 0306 : comme précédemment.
0307 : aucun record du set spécifié ne répond aux critères définis.
0317 : le record spécifié par current n'existe plus dans la base de données.
0304 : les data-items spécifics n'existent pas dans le sous-shéma pour le record spécifié par current.

+ W A R N I N G : 0350 : comme précédemment.

- Si les data-items spécifiés dans ces deux derniers ordres FIND sont déclarés SEARCH KEYS, ces ordres sont des ordres d'accès direct pour un set.

2.3. Accès aux records d'une area

- FIND { NEXT | PRIOR | FIRST | LAST | n | VAR } ⟨nom de type de record⟩ RECORD OF ⟨nom d'area⟩ AREA.

Le current de l'area concernée contiendra la DBK d'un record de cette area si la clause NEXT ou PRIOR est présente. Les records d'une area étant ordonnés par ordre croissant de valeur de leur

DBK, on accède donc au premier (si FIRST), au dernier (LAST), au suivant ou précédent le record spécifié par le current (si NEXT ou PRIOR) ou au $i^{ème}$ (si n ou VAR) record de l'area concernée.
Si la clause <nom de type de record> est présente, on n'accède qu'aux records de ce type.

- Valeurs de ERROR-STATUS :

+ E R R O R : 0301,0318,0310 : comme précédemment.
0307 : aucun record ne possède une valeur de DBK supérieure (NEXT) ou inférieure (PRIOR) à celle spécifiée par current.
0326 : l'area est vide ou ne contient aucun record du type spécifié.
0307 : i est supérieur au nombre de records de l'area.
0302 : le record spécifié par current n'existe pas pour l'area.

+ W A R N I N G : 0351 : comme précédemment.

- FIND <nom de type de record> RECORD
Cet ordre permet l'accès à des record pour lesquels sa clause LOCATION MODE est CALC.
On accède alors au premier record d'une area pour lequel les valeurs de CALC KEYS sont données dans les zones de UWA correspondant à ces data-items.
L'area est spécifiée par la zone de UWA associée à la clause AREA-ID si la clause WITHIN associée à ce type record comporte plus d'un nom d'area.

- Valeurs de ERROR-STATUS
Code d'erreur : 0301, 0318, 0310 : comme précédemment.
0323 : le nom de l'area spécifiée par AREA-ID n'appartient pas à la clause WITHIN correspondante.
0326 : aucun record de l'area spécifié ne possède de telles valeurs de CALC KEYS.

- FIND NEXT DUPLICATES WITHIN < nom record > RECORD
Cet ordre permet l'accès à des records dont le LOCATION MODE est CALC.

Le current du Run-unit contiendra la DBK d'un record de type spécifié et possédant comme valeurs de CALC KEYS, celles présentes en UWA.
On accède au premier record d'une area (à partir du record référencé par current) qui possède les mêmes valeurs de CALC KEYS que celles présentes en UWA.
Le nom de l'area est spécifié par AREA-ID si c'est nécessaire.
Si le current référence un record dont les valeurs de CALC KEYS ne sont pas égales aux valeurs de UWA, l'ordre a le même effet que l'ordre précédent.

- Valeurs de ERROR-STATUS

+ E R R O R : 0301, 0318, 0310, 0323 : comme précédemment.
0326 : aucun record ne peut être ainsi sélectionné.

+ W A R N I N G : 0350, 0351 : comme précédemment.

2.4. Ordre GET

L'ordre GET CODASYL met les valeurs de data-items associées à un record dans la zone de communication :

Syntaxe

GET [<nom de type de record>] .
ou
GET <nom de type de record> ; <nom data-item1 > , < nom data-item2>,...

Le current du run-unit contiendra la DBK du record concerné.
Dans sa première forme, l'ordre GET donne en UWA toutes les valeurs associées au record, dans sa deuxième il ne donne que les valeurs des data-items spécifiés.

3. Justification de l'ordre DML'

Clause FROM

Un système CODASYL ne permet de travailler qu'avec une seule base de données, il est donc superflu de permettre à la clause FROM de citer une étiquette relative à un ordre OPEN; c'est pourquoi cette clause n'a de raisons d'être qu'avec la clause VIA.

Clause IF

La condition <condition> portera sur des data-items du type de record concerné.
Certaines formes de la condition permettront un accès "direct".
On considère qu'une condition permet de générer des ordres d'accès "direct" : pour un set :
lorsque le type de record member du type de set concerné possède des data-items déclarés SEARCH KEY (soit : data-item1, data-item2, ...) et que chacun de ces data-items se retrouve dans la condition sous la forme : ...AND data-item1 = val1 AND...AND data-item2 = val2 AND...

pour une area :

a.) Lorsque le type de record concerné par l'ordre REACH possède un location mode CALC et que chacun des data-items déclarés CALC KEY (soit data-item3, data-item4, ...) se retrouvent dans la condition sous la forme :
... AND data-item3 = val3 AND ... AND data-item4 = val4 AND...

b.) Lorsque le data-item : ID-AREA se retrouve dans la condition sous la forme :
... AND ID-AREA = val AND ...
ou
... AND ID-AREA NOT = val AND ...
Cette dernière condition ne constitue pas un accès direct au sens propre mais elle permet de rentabiliser le nombre d'accès en limitant la clause WITHIN à certains noms d'areas présents (=) ou absents (not=) dans la condition.

Quelle forme générale adopter pour la condition ?
Il apparaît que la forme la plus intéressante est le produit de critères simples car elle permet au compilateur de détecter aisément quelle forme particulière de la condition permet l'accès "direct".

Des formes complexes telles que :

- produit de sommes de critères simples,
- somme de produits de critères simples,

multiplient très rapidement le nombre de conditions possibles portant sur des CALC KEYS, SEARCH KEYS ou le data-item ID-AREA et alourdiront le travail du compilateur.
De plus, toute condition ou partie de celle-ci ne permettant pas d'accès "direct" devra être transformée en ordre IF COBOL qui sera exécuté sur base des valeurs de data-items présentes dans la zone de communication.
Tout ordre REACH avec une condition ne permettant pas l'accès "direct" n'est donc pas différent d'un même ordre REACH (sans condition) suivi d'un ordre IF COBOL, c'est pourquoi la forme générale de la condition pour l'ordre DML' est celle orientée accès "direct", c-à-d :

- critère simple,
- produit logique de critères simples.

Remarquons qu'un critère simple avec une ⟨liste des valeurs⟩ introduit indirectement la forme somme de critères simples dans le produit car :
⟨nom data-item⟩ = (⟨liste des.valeur⟩) s'interprète comme
(⟨nom data-item⟩ = val1 OR ⟨nom data-item⟩ = val2 ...)
val1, val2 étant les valeurs de la liste.

Clause GET

Cette clause est simplifiée par rapport à la forme SPHINX.
Cette simplification est essentiellement due au fait que un ordre REACH avec clause IF nécessite que toutes les valeurs de data-item de la condition soient présentes dans la zone de communication (sauf les CALC KEYS, SEARCH KEYS et ID-AREA). Une clause GET globale évite ainsi les problèmes d'incohérence entre une clause IF et GET avec liste de data-items.

Valeurs de S-ERROR

Ces valeurs sont fonction des valeurs de ERROR-STATUS relatives aux ordres d'accès CODASYL qui serviront à la génération d'un texte équivalent à une boucle d'accès.
Ces ordres diffèrent selon les cas d'accès possibles pour un ordre REACH.

Nous distinguons :

pour le format 1 : - l'accès aux records d'un set (owner ou member).
- l'accès aux records d'un type donné; cet accès est alors basé sur les areas de la clause WITHIN de ce type de record.
- si l'ordre possède en plus une clause IF, on distingue, pour les accès aux records members, ceux basés sur les valeurs de SEARCH KEYS ou non et l'accès par areas peut être limité aux areas déterminées par une condition portant sur ID-AREA et/ou être basé sur des valeurs de CALC KEYS.

pour le format 2 : - l'accès sera basé sur une valeur de DBK fournie par une variable de travail.

La structure détaillée d'un texte généré dans chacun des cas se trouve décrite dans le chapitre cinq. Retenons toutefois les principes qui ont permis la définition des valeurs de S-ERROR :

- Lors de l'accès à un set, il faut préalablement positionner son current avec la DBK du record relatif à ⟨étiq2⟩ , cette opération peut engendrer les codes 0203, 0206.
  L'accès au record OWNER peut provoquer le positionnement des valeurs 0203, 0206, 0207, 0208.
  L'accès aux records members, généré par un ordre FIND NEXT ... et un ordre FIND ... USING de repositionnement de current avec la DBK de la réalisation courante avant une itération, peut amener les valeurs 0201, 0206, 0207, 0208.
- L'accès aux records des areas se fera indépendemment pour chacune d'elle, les erreurs détectées lors de l'examen d'une area n'empêchent pas l'exécution pour les autres areas d'une clause WITHIN; c'est pourquoi ces erreurs ne constituent qu'un code d'avertissement et peuvent générer les valeurs 0209, 0210, 0211.
- L'accès basé sur la DBK fournie par une variable de travail sera généré par l'ordre :

  FIND ... USING ⟨var. travail⟩

  et les valeurs de S-ERROR qui lui sontassociées découlent directement des valeurs ERROR-STATUS pour un tel ordre.
- Certaines valeurs de ERROR-STATUS telles que celles indiquant une fin de set, d'area... interviennent dans le texte généré

comme valeur de contrôle d'itérations et provoquent soit la la sortie de boucle, soit un accès basé sur de nouveaux critères.

END

Syntaxe

@ END [⟨étiq⟩].

Règles syntaxiques

1.- ⟨étiq⟩ : est le nom d'une étiquette connue jusqu'à cet endroit et définie dans un ordre REACH ou PREPARE.

2.- Il ne correspond qu'un seul ordre END à tout ordre REACH ou PREPARE.

3.- Si la clause ⟨étiq⟩ est absente, l'ordre END se réfère implicitement au dernier contexte ouvert qui n'a pas encore été clôturé à cet endroit, celui-ci ne peut avoir été ouvert par un ordre OPEN.

Sémantique

1.- L'ordre END définit la fin du contexte ouvert par un ordre REACH ou PREPARE. Après son exécution, l'étiquette du contexte et la réalisation courante qui lui étaient éventuellement associés sont rendues indisponibles.
L'ordre END clôture également tous les contextes qui lui sont emboîtés et qui n'auraient pas encore été fermés à cet endroit; ces contextes ont été ouverts par des ordres REACH ou PREPARE.

2.- Lorsqu'un ordre END implicite ou explicite est relatif à un ordre REACH, son exécution commande l'itération automatique de la boucle REACH- END; les instructions DML' et COBOL situées entre l'ordre REACH et l'ordre END qui lui correspond constituent le corps de la boucle et seront exécutées pour chaque réalisation courante de cette boucle.

NEXT

Syntaxe

@ NEXT [⟨étiq⟩][.]

Règles syntaxiques

1.- Cet ordre doit se trouver dans le contexte ⟨étiq⟩ ;
⟨étiq⟩ est le nom d'une étiquette définie par un ordre REACH.
2.- Si la clause ⟨étiq⟩ est absente, l'ordre NEXT se réfère implicitement à son contexte de base.
3.- Un ordre NEXT peut constituer la dernière instruction d'une branche TRUE ou FALSE d'une alternative IF COBOL.

Sémantique

1.- Si l'ordre NEXT est relatif à sa boucle de base, son exécution provoque l'abandon de la réalisation courante de cette boucle puis l'itération de cette dernière, la totalité de la boucle est exécutée pour la réalisation suivante si elle existe.
2.- Si l'ordre EXIT n'est pas relatif à sa boucle de base, son exécution provoque d'abord la sortie prématurée de tous les blocs, actifs à cet endroit et relatifs à un ordre REACH ou PREPARE, emboîtés dans le contexte ⟨étiq⟩ et ensuite l'itération forcée de cette boucle.
L'effet de l'ordre NEXT est donc équivalent à l'effet d'un ordre EXIT pour tous les blocs emboîtés dans le contexte ⟨étiq⟩ .

EXIT

Syntaxe

@ EXIT [⟨étiq⟩] .

Règles syntaxiques

1.- Cet ordre doit se trouver dans le contexte ⟨étiq⟩ .
2.- ⟨étiq⟩ peut avoir été défini par un ordre OPEN de format 1, un ordre REACH ou PREPARE.
3.- Si ⟨étiq⟩ est absente, on se réfère implicitement au contexte de base.
4.- Un ordre EXIT peut constituer la dernière instruction d'une branche TRUE ou FALSE d'une alternative IF COBOL.

Sémantique

1.- L'ordre EXIT provoque lors de son exécution la sortie prématurée du contexte auquel il fait référence.
2.- L'exécution du programme se poursuit à la première instruction (explicite ou implicite) qui suit l'ordre END ou l'ordre CLOSE du contexte que l'on quitte.
3.- Si l'ordre EXIT n'est pas relatif à son contexte de base, il provoque préalablement à l'effet décrit ci-dessus, l'abandon de tous les contextes actifs, relatifs à un ordre REACH ou PREPARE, emboîtés dans le contexte ⟨étiq⟩ .
4.- Si l'ordre EXIT est relatif à un ordre REACH, il provoque l'arrêt prématuré des itérations de cette boucle d'accès ainsi que de toutes celles qui lui seraient emboitées et encore actives à cet endroit.

SAVE

Cet ordre affecte à une variable de travail, une valeur qui est la référence à cette réalisation.

1. Ordre DML'

Syntaxe

@ SAVE [⟨étiq⟩] IN ⟨ Var.trav.⟩ .

Règles syntaxiques

1.- Cet ordre doit se trouver dans le contexte ⟨étiq⟩ .

2.- ⟨étiq⟩ est le nom d'une étiquette définie dans un ordre REACH ou PREPARE suivi d'un CREATE; ce nom ne peut être "IN".

3.- Si la clause ⟨étiq⟩ est absente, l'ordre SAVE se réfère implicitement à son contexte de base.

4.- ⟨Var. trav.⟩ est le nom d'une variable COBOL de type DATABASE-KEY.

2. Justification de l'ordre DML'

L'absence de valeurs pour S-ERROR se justifie par l'ordre qui sera généré : MOVE D.B.K. OF ⟨étiq⟩ TO ⟨Var. trav.⟩.
Les contrôles à l'exécution se feront donc comme pour un ordre COBOL.

CREATE

Cet ordre permet de créer un record et de l'insérer automatiquement dans des types de sets pour lesquels son type de record est automaticmember; en outre un set vide sera créé pour chaque type de set pour lequel il est déclaré OWNER.
Préalablement à un tel ordre, le programme doit fournir dans la zone de communication, les valeurs associées au record à créer.
Il doit pour ce faire réserver de la place en ZC pour ce record; ceci se fera grâce au bloc PREPARE-END.

1. Ordre DML'

Syntaxe

@ CREATE [⟨étiq1⟩] [, ⟨nom type de set⟩ : ⟨étiq2⟩]n o

Règles syntaxiques

1.- ⟨étiq1⟩ : est le nom d'une étiquette définie dans un ordre PREPARE et doit être celui du bloc de base.
2.- ⟨étiq2⟩ : est le nom d'une étiquette définie dans un ordre REACH ou PREPARE suivi d'un CREATE.
3.- L'étiquette du contexte de base ne peut apparaître dans la clause ⟨étiq2⟩ .
4.- L'ordre CREATE doit se trouver dans tous les contextes ⟨étiq2⟩
5.- Cet ordre doit être le premier ordre DML' du bloc PREPARE-END de base.
6.- Les types de sets spécifiés doivent être déclarés dans le sous-shéma et doivent avoir pour type de record :
   - OWNER : le type de record référencé par ⟨étiq2⟩ .
   - MEMBER : le type de record référencé par ⟨étiq1⟩ .
7.- n est au moins égal au nombre de sets pour lesquels le record à créer est automatic member.
8.- n peut au maximum être égal à ce nombre plus un :
   il se peut que la localisation de ce record se fasse par identification du record OWNER d'un set pour lequel il est manual member (contrainte du système CODASYL) et le record OWNER sera également identifié par une clause : ⟨nom type de set⟩ : ⟨étiq2⟩ ; le record OWNER ne sera toutefois pas inséré dans ce set.

9.- L'identification d'un set spécifié dans cet ordre peut nécessiter l'accès à des records qui ne sont pas référencés par l'ordre CREATE (contrainte du système CODASYL) et les areas contenant ces records doivent faire l'objet d'un ordre OPEN dans ce programme.

10.- Le type de record ne peut être déclaré avec une clause RETRIEVAL ONLY.

Règles sémantiques

1.- Le data-item ID-AREA attaché à ⟨étiq1⟩ doit contenir un nom d'area appartenant à une liste d'areas accompagnant la clause VALUE de ce data-item, si cette clause ne comporte qu'un seul nom d'area il prend cette valeur par défaut.

2.- Lorsque l'ordre PREPARE associé ne comporte pas de clause GET, les valeurs de data-items associées seront nulles (o ou ♭) et il en sera de même pour ceux non déclarés dans le sous-shéma.

Valeurs de S-ERROR

Code d'erreur

0401 : ⟨étiq2⟩ identifie un record inexistant ou n'existant plus.
0402 : les areas nécessaires à l'identification d'un set ne sont pas toutes ouvertes.
0403 : ou sous contrôle exclusif d'un autre programme ou OFF-line.
0404 : les sets existants dans la base de données ne permettent pas d'obtenir tous les records nécessaires à l'identification d'un set (contrainte du système CODASYL).
0405 : l'area dans laquelle le record doit s'insérer n'est pas ouverte.
0406 : les records référencés par ⟨étiq1⟩ et ⟨étiq2⟩ ne se trouvent pas tous dans des areas ouvertes avec un usage mode autre que RETRIEVAL.
0407 : il n'existe plus de place disponible dans l'area spécifiée par ID-AREA.
0408 : le nom d'area spécifié par ID-AREA n'est pas valide.
0409 : ou incompatible avec les valeurs associées au record à créer.

0410 : les valeurs des data-items déclarés IDENTIFIER ne sont pas uniques dans la base de données.
0411 : le programme n'a pas satisfait aux LOCKS.
0412 : après application d'une clause CHECK, certaines valeurs de data-items sont déclarées invalides.
0413 : les valeurs des data-items sont telles que la convertion dans le format du shéma est impossible.

2. Ordre CODASYL

L'ordre STORE crée un record et l'insère automatiquement dans les sets pour lesquels il est automatic member; le DBMS crée en outre un set vide pour chaque type de set pour lequel il est déclaré OWNER.

Syntaxe

STORE <nom type de record> SUPPRESS...

Règles syntaxique

Le sous-shéma du programme doit déclarer le type de record spécifié ainsi que les data-items spécifiés dans son location mode, au moins une area de sa clause WITHIN, tous les types de sets pour lesquels il est automatic member et tous les data-items spécifiés ou référencés dans les SOS associées à ces types de sets ainsi que ses data-items ASCENDING/DESCENDING KEY pour ces types de sets.

Règles sémantiques

1.- Une DBK sera associée au record à créer sur base de sa description dans le sous-shéma et des valeurs fournies par le programme
2.- Le programme doit assurer la présence des valeurs suivantes :
    a.- Les data-items spécifiés dans le location mode doivent être initialisés en UWA.
    b.- Les sets pour lesquels le record est automatic member seront identifiés par valeurs des data-items et currents référencés dans la SOS de chacun des sets concernés. Les valeurs des data-items seront donc présentes en UWA et les currents seront éventuellement mis à jour.

3.- Pour chaque location mode impliqué dans l'exécution d'un tel ordre :

a.- s'il est direct, la zone spécifiée par cette clause contiendra une DBK ou sera nulle (uniquement si elle concerne le record à créer).
Si la clause WITHIN associée comporte plus d'un nom d'area la zone associée à la clause AREA-ID éventuelle sera initialisée avec un nom d'area présente dans la clause WITHIN et déclarée par le sous-shéma.

b.- s'il est CALC, les data-items spécifiés dans cette clause doivent être initialisés en UWA ainsi que la zone associée à la clause AREA-ID si c'est nécessaire.

c.- s'il est VIA, les data-items associés à la SOS du type de set spécifié doivent être initialisés ainsi que la zone associée à la clause AREA-ID si c'est nécessaire. Le record à créer sera alors inséré le plus près possible de son point logique d'insertion dans le set.

d.- si aucune clause location mode n'est spécifiée, la DBMS insère le record dans l'area spécifiée par les clauses WITHIN et AREA-ID et lui assigne une DBK arbitraire dans l'area concernée.

4.- Les data-items associés à ce record et qui ne sont pas déclarés dans le sous-shéma auront une valeur nulle.

5.- La clause SUPPRESS empêche la mise à jour de certains currents après l'exécution de cet ordre.

Valeurs de ERROR-STATUS

E R R O R

1201 : l'area dans laquelle le record doit s'insérer n'est pas ouverte.

1221 : un record impliqué dans cet ordre se trouve dans une area qui n'est pas ouverte.

1209 : les records impliqués dans cet ordre ne se trouvent pas tous dans des areas ouvertes avec un usage mode autre que retrieval (uniquement pour les records susceptibles de subir des modifications c-à-d, les records OWNER immédiats du record à créer dans un chemin d'accès défini par une SOS).

1218 : certaines areas nécessaires à l'exécution de cet ordre se trouvent sous contrôle exclusif d'un autre programme.

1212 : le DBMS ne dispose plus de DBK disponible pour l'area spécifiée.

1202 : la DBK fournie ou obtenue à partir des valeurs de CALC KEYS n'est pas compatible avec l'area spécifiée.

1205 : le record viole une clause DUPLICATES NOT ALLOWED.

1225 : lors de l'identification d'un set par SOS, les critères d'identification ne permettent pas de sélectionner un set existant.

1210 : le programme n'a pas satisfait les PRIVACY LOCKS.

1227 : après application d'une clause CHECK, certaines valeurs de data-items sont déclarées invalides.

1223 : lors de l'identification d'un record dans une area par clause AREA-ID, le nom spécifié indique une area n'existant pas pour la clause WITHIN.

1219 : les valeurs des data-items en UWA sont telles qu'elles ne peuvent être converties dans le format spécifié dans le shéma.

3. Justification de l'ordre DML'

Un programme DML' ignore les clauses LOCATION MODE, WITHIN, AREA-ID, SET OCCURENCE SELECTION, c'est pourquoi les règles CODASYL associées à ces clauses n'existent pas dans l'ordre DML'. Remarquons que les règles relatives aux clauses WITHIN et AREA-ID se retrouvent dans celles associées au data-item ID-AREA de l'ordre CREATE .
Toutefois le programmeur DML' ne peut ignorer totalement les notions de location mode et set occurence selection car elles impliquent certaines actions de sa part.
Afin de justifier certaines règles, nous présentons comment le compilateur prend en charge ces deux notions.

a.- Location mode

- S'il est direct, le programme ne doit fournir aucun renseignement.

  Exemple : RECORD R1
  LOCATION MODE IS DIRECT ⟨nom zone⟩ .

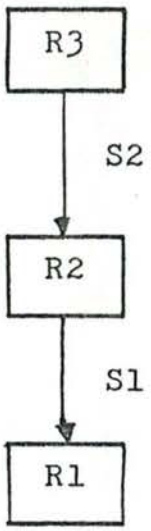
  Lors de la création d'un record de ce type, la DBK à fournir peut être nulle et le DBMS choisit la DBK à assigner au record en fonction du nom d'area fourni par clause WITHIN ou AREA-ID.
  Le compilateur génère alors :
  MOVE ZEROES TO ⟨nom zone⟩.
- S'il est CALC, les CALC KEYS doivent être initialisées mais l'ordre DML' n'en fait pas une règle ; si les CALC KEYS ne sont pas initialisées elles prendront par défaut la valeur nulle (ϕ ou 0) et le test de validité de ces valeurs se fera à l'exécution de l'ordre CREATE (0409, 0410).
- S'il est VIA, le record à créer sera localisé dans une area en tant que member d'un set dont le type est spécifié dans cet ordre. Ce set est identifié par SOS et le programme doit fournir l'OWNER immédiat du record dans le chemin d'accès défini par cette SOS. Le compilateur doit vérifier si ce record OWNER est fourni par l'ordre DML'. Si le record à insérer est automatic member de ce set, le contrôle se fera par vérification de la règle syntaxique 6 et le cas où il est manual member justifie la règle 7.

b.- Set Occurence Sélection

Cette notion intervient pour l'identification des sets pour lesquels le record à créer est automatic member.
Pour chacun des sets ainsi concernés, le compilateur génère une série d'ordres destinés à reconstituer le chemin d'accès défini par sa SOS.
Le programme DML' fournissant le moyen d'identifier la première étape du chemin d'accès défini par SOS (l'étiquette fournie pour chaque set représente l'OWNER immédiat du record à créer), le compilateur dispose des moyens suffisants pour reconstituer les autres étapes du chemin.

Exemple :

Soit un record R1 à créer, son type de record est automatic member d'un type de set S1. La SOS associée à ce type de set est la suivante :



Le programme DML' fournit l'OWNER immédiat de R1 soit R2.
La SOS indique que ce record R2 sera identifié comme member d'un set de type S2 dont l'OWNER est identifié par current.
Le compilateur doit générer pour cet exemple :

FIND ⟨nom type de record relatif à R2⟩ USING D-B-K OF
⟨étiquette associée à R2⟩ .

MOVE d-itl OF ⟨étiquette relative à R2⟩ TO d-itl OF
⟨nom type de record relatif à R2⟩ .

MOVE d-it2 OF ⟨étiquette relative à R2⟩ TO d-it2 OF
⟨nom type de record relatif à R2⟩ .

FIND OWNER RECORD OF S2 SET.

Ces ordres CODASYL doivent être exécutables et par conséquent l'area contenant R3 doit être ouverte et disponible pour le programme.
Les ordres COBOL (MOVE) doivent également être possibles et l'ordre REACH donnant accès aur record R2 doit s'accompagner d'une clause GET.
Les ordres qui seront àinsi générés justifient les règles syntaxiques 8 et 9.

Les valeurs de ERROR-STATUS 0401, 0402, 0403, 0404 proviennent de l'exécution des ordres CODASYL générés pour la reconstitution d'une SOS :

0401 = ERROR-STATUS = 0326
0402 = " " = 0301
0403 = " " = 0318
0404 = " " = 0322

Les valeurs de ERROR-STATUS résultant de l'ordre STORE et concernant une SOS n'interviennent donc pas pour S-ERROR, c-à-d : 1221,1218, 1225 .
Toutes les autres valeurs de S-ERROR sont directement liées aux valeurs de ERROR-STATUS.

PREPARE

Syntaxe

@ PREPARE ⟨nom type de record⟩ = ⟨étiq⟩ [GET] .

Règles syntaxiques

1.- ⟨nom type de record⟩ est le nom d'un type de record déclaré dans le sous-shéma du programme.
2.- ⟨étiq⟩ est le nom d'une étiquette qui sera attachée au bloc PREPARE- END défini; ce nom doit être unique.
3.- A tout ordre PREPARE doit correspondre un ordre END explicite ou non qui clôture son contexte.
4.- Entre un ordre PREPARE et certains ordres lui faisant référence (MODIFY, REACH, ATTACH, DETACH, TRANSFER, SUPPRESS, CREATE non attachés à ce PREPARE), doit se trouver un ordre CREATE qui lui fait référence.
5.- Dans le contexte d'un PREPARE doit se trouver un ordre CREATE qui lui fait référence.

1.3. Règles sémantiques

1.- L'ordre PREPARE définit un contexte qui s'étendra jusqu'au END qui le clôture en aval dans le programme; à ce contexte est associée une étiquette de nom ⟨étiq⟩ qui pourra dans la suite être affectée à une réalisation du record qui sera créé par un ordre CREATE situé dans le contexte ⟨étiq⟩ .
2.- Après l'exécution du PREPARE, l'étiquette ⟨étiq⟩ est connue dans le contexte défini mais non affectée.
La clause GET réserve dans la zone de communication de l'espace destiné à contenir les valeurs des data-items associées au record à créer; cet espace est lié au contexte ⟨étiq⟩ .
3.- Le data-item ID-AREA est présent dans la zone ZC dans tous les cas (avec ou sans clause GET).
4.- L'ordre PREPARE initialise les zones de ZC réservées.
Cette initialisation consiste à donner une valeur nulle aux data-items (ϕ pour valeurs alphabétiques ou alpha-numériques,
o pour les valeurs numériques ou de type DBK).

SUPPRESS

Cet ordre supprime un record de la base de données ainsi que certains autres records qui lui sont attachés.

1. Ordre DML'

Syntaxe

@ SUPPRESS [⟨étiq⟩] [ONLY | SELECTIVE | ALL] {NEXT. | EXIT.}

Règles syntaxiques

1.- Cet ordre doit se trouver dans le contexte ⟨étiq⟩ .

2.- ⟨étiq⟩ est le nom d'une étiquette définie dans un ordre REACH ou PREPARE suivi d'un CREATE. Ce nom ne peut être ONLY, SELECTIVE, ALL , NEXT ou EXIT.

3.- Si la clause ⟨étiq⟩ est absente, l'ordre SUPPRESS se réfère implicitement à son contexte de base.

4.- Le bloc auquel se réfère l'ordre SUPPRESS ne peut se trouver dans aucun contexte auquel serait associée une réalisation courante dont le record asscocié risque d'être supprimé (selon la clause ONLY, SELECTIVE, ALL).

Règles sémantiques

1.- Cet ordre détache le record référencé par ⟨étiq⟩ de tous les sets pour lequel il est member et le supprime ensuite de la base de données.

2.- Cette suppression entraîne la suppression d'autres records qui sont fonction de la clause :

- ONLY : tous les records members mandatory font également l'objet d'un ordre DELETE ONLY.
- SELECTIVE : tous ses records members mandatory ou optional (uniquement s'ils ne participent à aucun autre set) font également l'objet d'un ordre DELETE SELECTIVE.
- ALL : tous ses records members font également l'objet d'un ordre DELETE ALL.

s'il ne s'accompagne d'aucune de ces clauses, l'ordre SUPPRESS ne s'applique qu'au record désigné par ⟨étiq⟩ et uniquement s'il n'est OWNER que sets vides.

3.- Quelle que soit l'issue de l'opération précédente, la clause NEXT ou EXIT est exécutée, provoquant impérativement soit l'itération forcée, soit l'abandon prématuré du bloc auquel se réfère l'ordre SUPPRESS.

Valeurs de S-ERROR

Code d'erreurs

0501 : le programme n'a pas fourni les locks nécessaires à la suppression de tous les records.
0502 : toutes les areas nécessaires ne sont pas ouvertes.
0503 : ou ne sont pas toutes ouvertes avec un usage mode autre que RETRIEVAL.
0504 : quelques areas sont sous contrôle exclusif d'un autre programme ou OFF-line.
0505 : l'ordre entraîne la modification d'un data-item ACTUAL RESULT et la clause CHECK appliquée à cette nouvelle valeur le déclare invalide.
0506 : le sous-shéma du programme ne déclare pas tous les records ou sets impliqués dans cet ordre.
0507 : l'ordre sans clause ONLY, SELECTIVE ou ALL s'applique à un record OWNER d'un (ou plusieurs) set(s) non vide(s).
0508 : le record référencé par ⟨étiq⟩ est inconnu.
0509 : ou dans une area qui n'est pas ouverte.

2. Ordre CODASYL

Syntaxe

DELETE ⟨nom type record⟩ [ONLY | SELECTIVE | ALL] .

Règles syntaxiques

Le type de record spécifié doit être déclaré dans le sous-shéma nommé dans le programme.

Règles sémantiques

1.- Le record à supprimer est référencé par le current du run-unit.
2.- Les records rattachés à ce record et qui seront également supprimés sont fonction des clauses ONLY, SELECTIVE, ALL; ces clauses ont la même signification que celle de l'ordre DML'.

3. Justification de l'ordre DML'

Les clauses NEXT et EXIT sont obligatoires compte tenu de la structure de bloc.

Règles syntaxiques

Elles sont directement liées à la structure de bloc d'un programme.

Règles sémantiques

Elles découlent naturellement de l'ordre CODASYL qui sera gégéré.

Seule la règle 3 est relative à l'ordre SHINX équivalent.

Valeurs de S-ERROR

0508 et 0509 : résultent de l'ordre FIND...USING qui sera généré conformément à la mise à jour des currents.
Les autres valeurs trouvent un équivalent CODASYL.

MODIFY

Cet ordre modifie toutes ou une partie des valeurs de data-items associées à un record.

1. Ordre DML'

Syntaxe

@ MODIFY [⟨étiq⟩ : ⟨data-itl⟩ , ⟨data-it2⟩ ,...]n o

Règles syntaxiques

1.- Cet ordre doit se trouver dans le(s) contexte(s) étiq .
2.- étiq est une étiquette définie par un ordre REACH ou PREPARE suivi d'un create et accompagnant d'une clause GET.
3.- Si une clause ⟨étiq⟩ est absente, l'ordre MODIFY se refère implicitement à son contexte de base.
4.- Toutes les clauses ⟨étiq⟩ sont différentes.
5.- Les différentes clauses ⟨étiq⟩ : ⟨data-itl⟩ ,... sont séparées par le signe ; .
6.- ⟨data-itl⟩ , ⟨data-it2⟩ doivent être des noms de data-items définis dans le sous-shéma du programme et appartennant au type de record référencé par ⟨étiq⟩ .
7.- Les data-items nommés ne peuvent être déclarés

{ACTUAL / VIRTUAL} {SOURCE / RESULT}

Règles sémantiques

1.- Pour chaque clause ⟨étiq⟩ , l'exécution de l'ordre MODIFY modifie la base de données en associant le record référencé par ⟨étiq⟩ aux nouvelles valeurs des data-items de ce record qui auront probablement été rangées dans la zone de communication affectée à ⟨étiq⟩ .
2.- Si ⟨data-itl⟩ , ⟨data-it2⟩ ... sont spécifiés, seuls les valeurs de ces data-items seront modifiées, sinon toutes les valeurs des data-items du record référencé par ⟨étiq⟩ seront modifiées (s'ils sont déclarés dans le sous-shéma et ne sont pas Actual ou Virtual source ou result).

3.- Une tentative de modification du data-item ID-AREA n'est pas prise en compte.

4.- Les modifications sont exécutées indépendemment des autres et une modification est exécutable si aucun des événements correspondant aux valeurs de S-ERROR n'est détecté.

Valeurs de S-ERROR

Code d'avertissement

0601 : le programme n'a pas satisfait aux LOCKS nécessaires pour une modification de cet ordre.

0602 : une clause CHECK appliquée lors d'une modification a révélé une nouvelle valeur de data-item invalide.

0603 : une area relative à une modification a été ouverte avec un usage mode RETRIEVAL.

0604 : la conversion entre formats du shéma et sous-shémas ne peut s'effectuer pour certaines valeurs de data-items.

0605 : une modification ne peut s'effectuer car le record concerné n'existe pas ou plus dans la base de données.

0606 : une modification d'un data-item déclaré IDENTIFIER lui attribue une valeur existant déjà pour un autre record de la base de données.

0607 : l'area nécessaire n'est pas ouverte.

S-ERROR tient compte de la dernière erreur détectée pour l'ordre MODIFY et le registre S-COUNT indique le nombre total d'erreurs détectées pour cet ordre.

2. Ordre CODASYL

L'ordre MODIFY CODASYL (sans clause USING) permet de modifier les valeurs de data-items d'un record.

Syntaxe

MODIFY { [⟨nom type de record⟩] .
⟨nom type de record⟩ ; ⟨data-it1⟩ , ⟨data-it2⟩ ,... }

Règles syntaxique

1.- Le type de record ainsi que les data-items spécifiés doivent être déclarés dans le sous-shéma.
2.- ⟨data-it1⟩ ,⟨data-it2⟩ ,... ne peuvent être définis comme ACTUAL/VIRTUAL SOURCE/RESULT sauf s'ils sont relatifs à des data-items d'un record OWNER et interviennent dans une SOS car alors ils peuvent modifier les étapes d'un chemin d'accès à un set.

Règles sémantiques

1.- Le record dont les valeurs sont à modifier est référencé par le current du run-unit et les nouvelles valeurs de data-items doivent se retrouver en UWA.
2.- Seuls seront modifiés les data-items déclarés dans le sous-shéma du programme.
3.- Lorsque les data-items àmodifier sont déclarés SEARCH KEY, CALC KEY, ASCENDING/DESCENDING KEY, le DBMS se charge d'effectuer les modifications nécessaires dans la base de données (réajustement d'index, modification de l'ordre des members d'un set,...) et la DBK du record concerné reste inchangée.

3. Justification de l'ordre DML'

Syntaxe

Par rapport à l'ordre SPHINX équivalent, l'ordre DML' permet de préciser outre l'étiquette relative au record à modifier, le nom des data-items à modifier.
Ces clauses supplémentaires sont nécessaires car un ordre REACH ou PREPARE DML' s'accompagne d'une clause GET globale et par conséquent le programmeur qui ne veut effectuer l'ordre MODIFY que pour certaines valeurs de data-items doit pouvoir le préciser au moment de l'écriture de cet ordre.

Règles syntaxiques

La règle 7 est imposée par le système CODASYL. Elle est toutefois plus restrictive que celle de l'ordre MODIFY dont elle s'inspire : on n'admet pas de modifier des valeurs de data-items SOURCE ou RESULT même s'ils sont relatifs à une SOS : dans ce cas l'effet de l'ordre est de modifier un chemin d'accès à un set et ceci ne répond plus à la fonction remplie par un ordre MODIFY DML' (une telle fonction est remplie par les ordres DML' ATTACH et DETACH ou TRANSFER).

Règles sémantiques

La règle 3 provient du fait que conformément à ce qui a été défini dans le DDL', le data-item ID-AREA n'existe pas dans la base de données.
La règle 4 provient de l'ordre SPHINX équivalent, elle n'existe que pour offrir au programmeur une facilité d'écriture. Un ordre MODIFY multiple correspond à une série d'ordre MODIFY simples successifs.

Valeurs de S-ERROR

La valeur 0605 provient de l'ordre FIND ... USING qui sera généré conformément à la mise à jour des currents.
Les autres valeurs trouvent un équivalent pour ERROR-STATUS.

ATTACH

Cet ordre rattache à un ou plusieurs types de set un record dont le type est déclaré optional ou mandatory manual.

1. Ordre DML'

Syntaxe

@ ATTACH [⟨étiq1⟩] [TO ⟨étiq2⟩ VIA ⟨nom type de set⟩]$_{1}^{n}$

Règles syntaxiques

- ⟨étiq1⟩ et ⟨étiq2⟩: sont des noms d'étiquettes connues jusqu'à cet endroit et définies dans un ordre REACH ou PREPARE suivi d'un ordre CREATE. Une clause ⟨étiq2⟩ doit être différente de ⟨étiq1⟩ .
- ⟨étiq1⟩: est relative à un record dont le type est, soit, R1.
- ⟨étiq2⟩: est relative à un record dont le type est, soit, R2.
- ⟨nom type de set⟩ : est le nom d'un type de set déclaré dans le sous-shéma du programme et pour lequel R1 est OPTIONAL ou MANDATORY MANUAL MEMBER et R2 est OWNER ou MEMBER.
  Un nom de set ne peut apparaître qu'une seule fois dans cet ordre.
- Cet ordre doit se trouver dans les contextes ⟨étiq1⟩ et ⟨étiq2⟩ .
- n réprésente le nombre max. de types de set déclarés dans le sous-shéma du programme et pour lesquels R1 est OPTIONAL ou MANDATORY MANUAL MEMBER.

Règles sémantiques

Chaque insertion se fera indépendemment des autres et une erreur détectée pour l'une d'elle positionne S-ERROR à une valeur (= code d'avertissement) et l'opération n'est pas exécutée.
S-ERROR rend compte de la dernière erreur détectée et S-COUNT donne le nombre total d'erreurs pour l'ordre ATTACH.

Valeurs de S-ERROR

Code d'erreur

0701 : le record référencé par ⟨étiq1⟩ n'existe pas ou plus.
0702 : ou se trouve dans une area qui n'est pas ouverte.
0703 : le programme n'a pas satisfait aux LOCKS nécessaires pour cet ordre.

Code d'avertissement

0704 : un record référencé par ⟨étiq2⟩ n'existe pas ou plus.
0705 : ou se trouve dans une area qui n'est pas ouverte.
0706 : le record à insérer n'est pas optional ou mandatory manual member pour un des types de set spécifiés.
0707 : l'insertion dans un type de set entraîne pour un data-item déclaré IDENTIFIER une valeur existant déjà pour un set.
0708 : le record à insérer est déjà member d'un des types de set spécifiés.
0709 : une ou plusieurs areas concernées ont un usage mode RETRIEVAL
0710 : l'exécution de l'ordre entraîne une nouvelle valeur d'un data-item VIRTUAL RESULT pour un OWNER et celle-ci est déclarée invalide après application de la clause CHECK.
0711 : un record spécifié par ⟨étiq2⟩ n'appartient plus au type de set spécifié, mais l'insertion peut s'exécuter.

2. Ordre CODASYL

L'ordre INSERT permet d'attacher un record à un ou plusieurs set(s).

Syntaxe

INSERT [⟨nom de type de record⟩] INTO ⟨nom type set1⟩, ⟨nom type set2⟩ ,...

Règles syntaxiques et sémantiques

- ⟨nom de type de record⟩ est un type de record member optional ou mandatory manual pour chaque type de set concerné par l'ordre.
- La DBK du record à insérer sera présente dans le current du RUN-UNIT.
- Pour chaque type de set, un set particulier sera identifié par le current de ce type de set qui contiendra la DBK d'un record OWNER ou MEMBER.

3. Justification de l'ordre DML'

L'ordre INSERT multiple ne sera pas utilisé pour la génération.

Un ordre ATTACH sera équivalent à une suite d'ordre INSERT simples; ceci permet d'exécuter les insertions indépendemment les unes des autres.

Les valeurs de S-ERROR : 0701, 0702, 0704, 0705 sont relatives aux ordres FIND ... USING générés conformément à la mise à jour des currents.

DETACH

Cet ordre détache un record de sets auxquels il est attaché et pour lesquels il est optional member.

1. Ordre DML'

Syntaxe

@ DETACH [⟨étiq⟩] [FROM ⟨nom type de set⟩]$_{1}^{n}$

Règles syntaxiques

1.- ⟨étiq⟩ est le nom d'une étiquette relative à un record, soit R1.

2.- ⟨nom type de set⟩ est le nom d'un type de set déclaré dans le sous-shéma et pour lequel ce type de record de R1 est optional member.

3.- Cet ordre doit se trouver dans le contexte ⟨étiq⟩ qui est relative à un ordre REACH ou PREPARE suivi d'un CREATE.

Règles sémantique

Pour chaque clause FROM l'ordre supprime l'appartenance du record référencé par ⟨étiq⟩ à un type de set spécifié. Chaque suppression est indépendante des autres et une erreur détectée pour l'une d'elle positionne un code d'avertissement et l'opération n'est pas exécutée.
S-ERROR rend compte de la dernière erreur détectée et S-COUNT indique le nombre total d'erreurs pour l'ordre.

Valeurs de S-ERROR

Code d'erreur

0801 : le record référencé par ⟨étiq⟩ n'existe pas ou plus.
0802 : ou se trouve dans une area qui n'est pas ouverte.
0803 : le programme n'a pas satisfait aux LOCKS nécessaires.

Code d'avertissement

0804 : le record à détacher n'est effectivement member d'un type de set spécifié.
0805 : le record à détacher n'est pas optional member pour un type de set spécifié.
0806 : les areas nécessaires à l'identification d'un set ne sont pas toutes ouvertes.
0807 : ou cetaines d'entre elles sont sous contrôle exclusif ou OFF-line.
0808 : ou ne sont pas toutes ouvertes avec un usage mode autre que RETRIEVAL.
0809 : une suppression entraîne la modification d'une valeur de data-item ACTUAL RESULT dans un record OWNER et la nouvelle valeur est déclarée invalide après application de la clause CHECK.

2. Ordre CODASYL

L'ordre REMOVE CODASYL permet de détacher un record d'un (ou plusieurs) set(s) au(x)quel(s) il appartient.

Syntaxe

REMOVE [⟨nom type de record⟩] FROM ⟨nom type set1⟩ , ⟨nom type de set2⟩ ,... .

Règles syntaxiques et sémantiques

Le record à détacher est optional member des sets spécifiés.
Le current du RUN-UNIT doit contenir la DBK du record à détacher.
Remarquons que pour cet ordre, le seul record à détacher suffit pour identifier un set parmis les types de set spécifiés, car un record n'est member que d'un seul set d'un type donné.

Justification de l'ordre DML'

Comme pour l'ordre ATTACH nous n'utiliserons pas la possibilité d'un ordre CODASYL multiple.
Un ordfe DETACH sera généré par une suite d'ordre REMOVE simples afin de rendre chaque opération indépendante des autres dans l'ordre. Les codes 0801, 0802 proviennent de l'ordre FIND ... USING qui sera généré conformément à la mise à jour des currents.

TRANSFER

Cet ordre est destiné à transférer l'appartenance d'un record member d'un set vers un autre set de même type. Seuls les records member ne pouvant faire l'objet d'ordre DETACH et ATTACH pourront faire l'objet d'un tel ordre c-à-d les members mandatory.
L'ordre regroupe alors deux fonctions : il détache un record d'un set auquel il appartient et l'insère dans un autre set identifié par l'ordre.

Syntaxe

@ TRANSFER ⟨étiq1⟩ [, TO ⟨étiq2⟩ VIA ⟨nom type de set⟩]$_{1}^{n}$

Règles syntaxiques

1.- ⟨étiq1⟩ et ⟨étiq2⟩ : sont des noms d'étiquettes relatives à des ordres REACH ou PREPARE suivis d'un CREATE.

2. Si une étiquette est absente, l'ordre désigne implicitement celle du bloc de base.

3.- Cet ordre doit se trouver dans tous les contextes ⟨étiq1⟩ et ⟨étiq2⟩ .

4.- ⟨nom type de set⟩ est le nom d'un type de set dont le type de record OWNER est celui référencé par ⟨étiq2⟩ et pour lequel le type de record relatif à ⟨étiq1⟩ est mandatory member.

5.- Une étiquette ⟨étiq2⟩ ne peut s'appeler "TO" ni "VIA".

6.- n représente au maximum le nombre de sets pour lesquels le type de record relatif à ⟨étiq1 est mandatory member.

7.- L'identification des sets précisés par ⟨étiq2⟩ peut impliquer d'autres records et sets que ceux précisés dans cet ordre ainsi que certaines valeurs de data-items relatifs à ceux-ci (contrainte du système CODASYL). Les types de ces records, sets ainsi que ces data-items doivent être déclarés dans le sous-shéma et éventuellement présents en ZC.

8.- Ils doivent également se trouver dans des areas ouvertes.

Règles sémantiques

Une clause TO ⟨étiq2⟩ VIA ... constitue un transfert partiel. Chaque transfert partiel est exécuté indépendemment des autres et une erreur détectée puur l'un d'eux positionne S-ERROR à un code d'avertissement et le transfert partiel n'est pas effectué. S-COUNT rend compte du nombre total d'erreurs détectées pour cet ordre.

Valeurs de S-ERROR

Code d'erreur

0901 : le record référencé par ⟨étiq1⟩ n'existe pas ou plus.
0902 : ou se trouve dans une area qui n'est pas ouverte.
0903 : le programme n'a pas satisfait aux LOCKS nécessaires.

Code d'avertissement

0904 : les areas contenant les records relatifs à ⟨étiq1⟩ et ⟨étiq2⟩ ne sont pas toutes ouvertes avec un usage mode autre que retrieval.
0905 : le record référencé par ⟨étiq1⟩ n'est pas effectivement member d'un type de set spécifié.
0906 : toutes les areas nécessaires à la localisation d'un set identifié par ⟨étiq2⟩ ne sont pas ouvertes.
0907 : ou certaines d'entre elles sont sous contrôle exclusif ou OFF-line.
0908 : la localisation d'un set ne peut se faire car les sets existant dans la base de données dans le sous-shéma sont insuffisants.
0909 : le record relatif à ⟨étiq2⟩ n'existe pas ou plus.

2. Ordre CODASYL

L'ordre MODIFY avec clause USING permet de modifier l'appartenance d'un record à un type de set.

Syntaxe

MODIFY ⟨nom type de record⟩ USING ⟨data-it1⟩ , ⟨data-it2⟩ ,...

Règles syntaxiques

1.- Le type de record spécifié et les data-items nommés (data-it1, data-it2,...) doivent être déclarés dans le sous-shéma.
2.- Les types de set et de record ainsi que les data-items implicitement référencés doivent être déclarés dans le sous-shéma.
3.- ⟨data-it1⟩ , ⟨data-it2⟩ ,... sont les noms de data-items permettant d'identifier le record OWNER immédiat du record à modifier dans un (ou plusieurs) chemin(s) d'accès à un (ou plusieurs) set(s) défini(s) par SOS.

Règles sémantiques

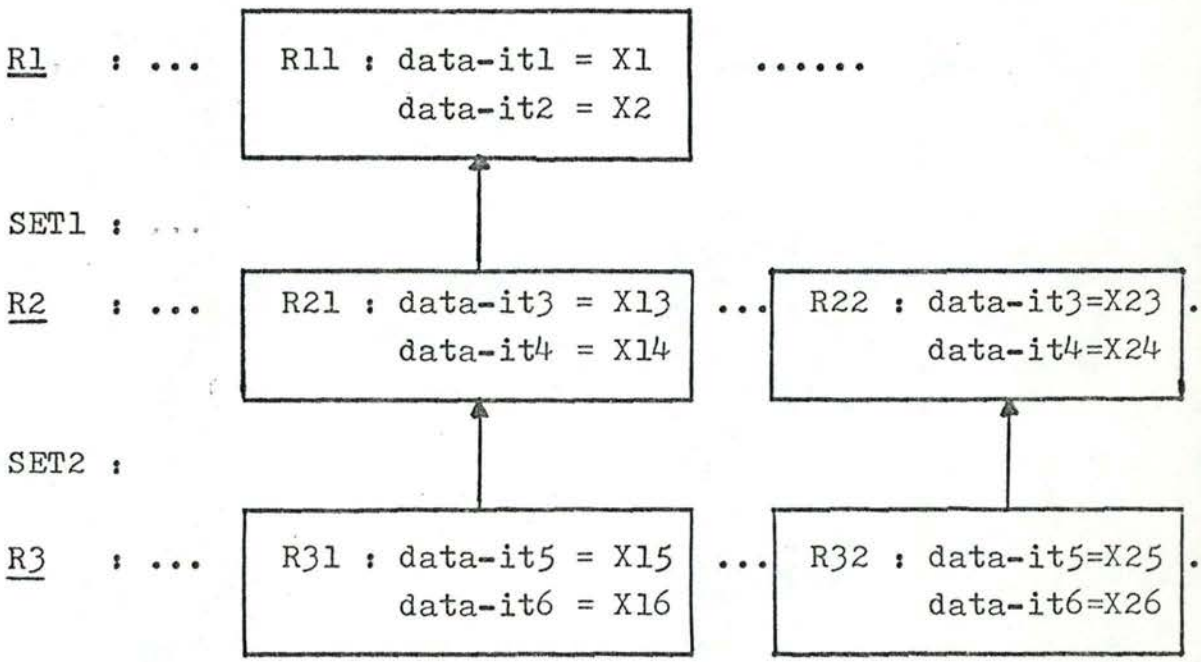
1.- Le record à modifier est identifié par current du run-unit.
2.- Pour chaque type de set implicitement référencé, le record à modifier sera transféré du set auquel il appartient vers un autre set identifié par les valeurs de ⟨data-it1⟩ ,... relatives à l'OWNER.
Pour chaque nouveau set, tous les data-items implicitement ou explicitement référencés par SOS doivent être initialisés en UWA, les currents concernés doivent être mis à jour ainsi que les zones associées à une clause ID-AREA relative à une SOS.

Exemple :

```
RECORD R1
RECORD R2 LOCATION MODE VIA Set2
RECORD R3 LOCATION MODE CALC USING data-it5, data-it6
          WITHIN area1, area2 AREA-ID Z1.
SET S1
    OWNER R2
    MEMBER MANDATORY R1
    SOS THRU LOCATION MODE OF OWNER
             USING data-it1, data-it2
SET S2
    OWNER R3
    MEMBER R2
    SOS THRU LOCATION MODE OF OWNER
             USING data-it3, data-it4
```

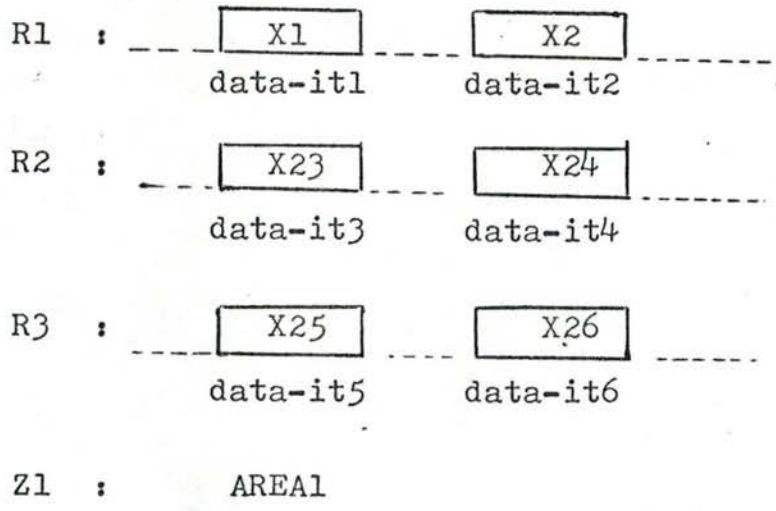
Ces clauses DDL impliquent que un set de type S1 sera identifié par identification d'un record OWNER de type R2 en tant que member d'un set de type S2 grâce à ses valeurs pour les data-items : data-it3 et data-it4 et dont l'OWNER est un record de type R3 identifié par ses valeurs de CALC-KEY : data-it 5, data-it6.
Dans un set de type S1, un record member est identifié par ses valeurs pour les data-items : data-it1, data-it2.
Supposons la situation suivante dans la base de données :

AREA 1



Un programme désirant modifier l'appartenance de R11 au type de set S1 en le rattachant à R22 devra effectuer les opérations suivantes:

1. Placer la DBK de R11 dans le current du RUN-UNIT.
2. Garnir les zones de UWA comme suit :



3. Exécuter l'ordre :

MODIFY R1 USING data-it1, data-it2, data-it3, data-it4.

Valeurs de ERROR-STATUS

0810 : Le programme n'a pas satisfait les privacy locks.

0804 : Les data-items spécifiés ne sont pas des data-items relatifs à une SOS d'un set pour lequel le record référencé par current est member.

0821 : Les records faisant l'objet d'une modification (le record référencé par current du run-unit, les records OWNERS immédiats de l'ancien et du nouveau chemin d'accès défini par SOS d'un type de set concerné) ne sont pas tous dans des areas ouvertes avec un usage mode autre que RETRIEVAL.

0809 : Les areas nécessaires à la localisation ne sont pas toutes ouvertes.

0818 : Les areas nécessaires à la localisation des records impliqués ne sont pas toutes disponibles (sous contrôle exclusif d'un autre programme ou OFF-line).

0823 : Une variable relative à une clause AREA-ID n'est pas initialisée.

0802 : ou la valeur fournie est incompatible avec les valeurs de CALC KEYS fournies.

0825 : Les valeurs fournies pour les data-items relatifs à une SOS ne permettent pas la sélection d'un set.

0822 : Le record identifié par current n'est pas member effectif des types de sets implicitement identifiés dans l'ordre.

3. Justification de l'ordre DML'

Les notions relatives à une SOS seront prises en charge par le compilateur comme pour l'ordre CREATE; on justifie ainsi les règles syntaxiques 7 et 8.

Cette reconstitution prend en charge les valeurs de ERROR-STATUS relatives à une SOS (0804, 0821, 0818, 0823, 0802, 0825) qui sont rendues au programme DML' sous la forme des codes 0906, 0907, 0908, 0909.

Les valeurs 0901, 0902 résultent de l'ordre FIND...USING généré conformément à la mise à jour du current du run-unit.

4.3.3.4. Les codes d'erreur grave

Nous ne pouvons pas énumérer tous les événements provoquant une anomalie grave car ils peuvent varier selon les DBMS CODASYL existant. Ils représentent les cas où le DBMS ne peut exécuter correctement un travail sur une base de données parce qu'ils affectent l'intégrité de la base de données ou celle du DBMS.

L'effet d'un tel événement positionne S-ERROR à un code d'erreur grave et l'exécution se poursuit à la première instruction suivant l'ordre CLOSE de forme 1 du programme.

Les cas d'erreurs graves sont par exemple :

- la base de données n'existe plus,
- erreur du système,
- manque de place mémorée,
- ....

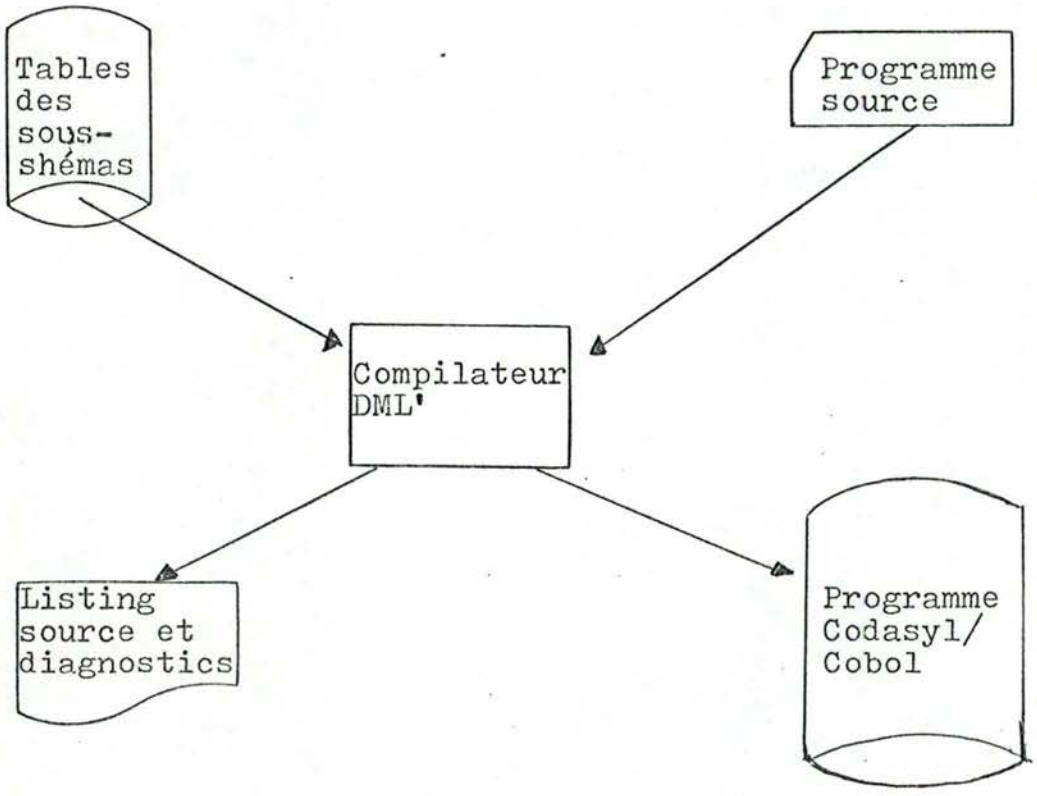
Chapitre 5 : Le compilateur DML'

Ce chapitre constitue la partie pratique du mémoire et présente l'analyse et la génération des ordres REACH, NEXT, EXIT, OPEN, CLOSE et END.
Il présente les tables et variables ainsi que les algorithmes décrivant les programmes qui constituent le compilateur.
Les programmes sont prévus pour le DBMS-20 de DIGITAL (DEC SYSTEM 2050) c'est pourquoi des notions générales analysées pour le système CODASYL n'apparaîtront pas dans ce chapitre.

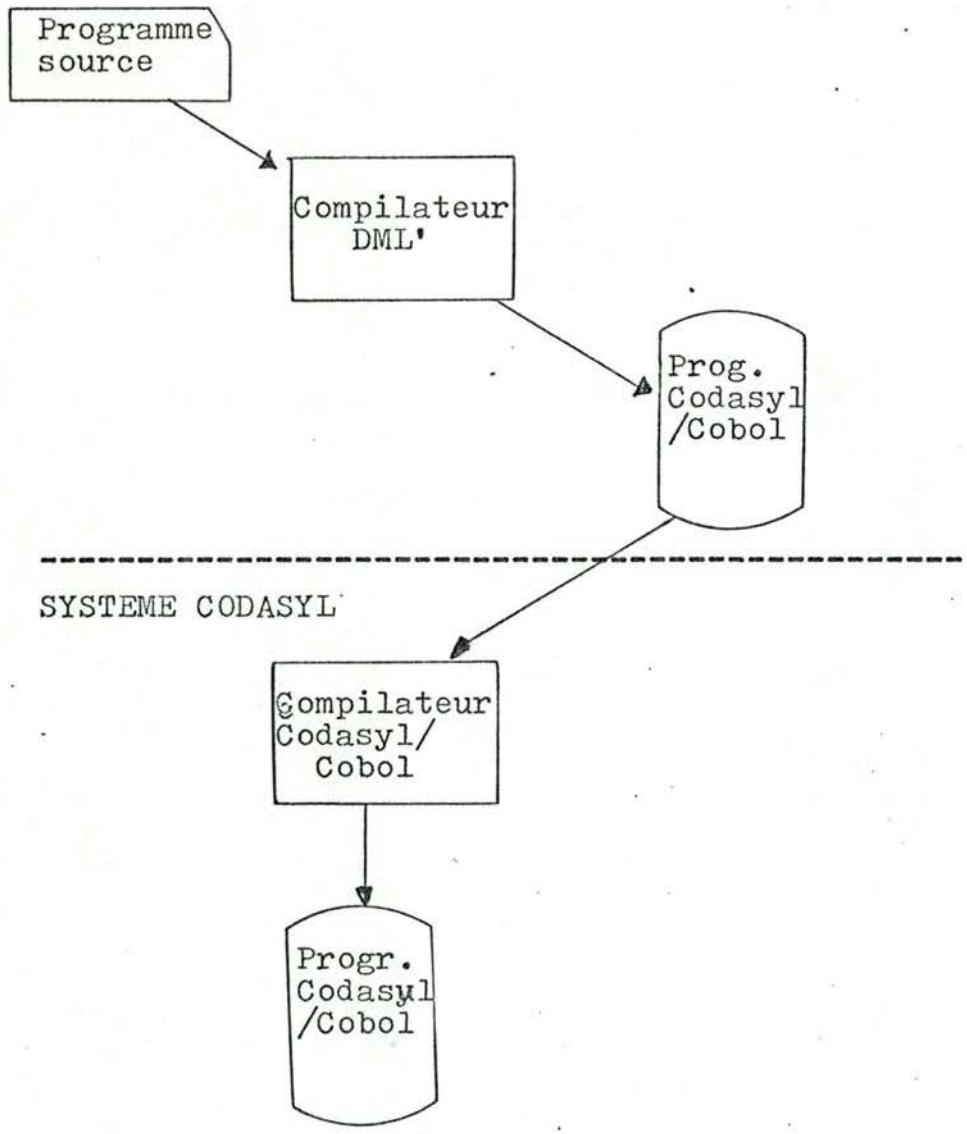
5.1. Les fonctions du compilateur DML'.

La compilation d'un programme réalise les fonctions suivantes :

- l'analyse d'un programme source DML'/Cobol,
- la génération d'un texte Codasyl/Cobol équivalent dans le cas où l'analyse reconnaît le programme correct,
- l'édition du programme source avec les messages du compilateur(diagnostics).



- Une compilation complète d'un programme DML' s'effectue en deux étapes dont l'enchaînement peut se représenter :



- Le travail du compilateur DML' se déroule en trois phases :
  - L'analyse syntaxique vérifie la syntaxe et ses règles. Cette vérification se fait entre autres à l'aide de la description du sous-shéma du programme à compiler.
    Lorsque l'analyse syntaxique d'un ordre s'est révélée correcte, le compilateur procède à la phase de génération.
  - La génération consiste à recopier l'ordre dans le texte Codasyl/cobol à générer en tant que commentaire et de le faire suivre des ordres Codasyl/Cobol équivalents.

La troisième étape produit le listing source du programme DML' ainsi que les messages du compilateur.

5.2. Structure d'un programme CODASYL/COBOL

Un programme DML'/COBOL doit comporter la structure suivante :

- DATA DIVISION.

  W.S.S.

- PROCEDURE DIVISION.

  ordres DML'/COBOL

Après compilation, ce même programme comportera en W.S.S., la zone de communication ZC qui comprendra les zones suivantes :

- les registres d'erreur : S-ERROR et S-COUNT
- la zone des données qui est fonction des ordres REACH et PREPARE écrits dans le programme.

Cette zone est organisée d'une manière particulière qui permet de profiter de la structure de blocs du programme DML' pour ne conserver en mémoire que les informations relatives aux réalisations actives à chaque endroit. Cette zone est organisée en pile.
Les gestion de cette pile est statique (fixée au stade de la compilation) à la fois en raison de la rigidité de la structure de blocs et des possibilités du COBOL (pas de gestion dynamique de la mémoire).
L'accès aux informations de la pile peut s'effectuer en COBOL grâce à la technique de construction suivante :

- on réserve pour la pile une zone de longueur fixe correspondant à la longueur maximum que peut atteindre cette pile dans le programme; cette zone porte le nom de S-STACK.
- chaque fois que de l'espace doit être réservé dans cette pile, une nouvelle découpe de cette dernière est générée (ordre REDEFINES); cette découpe contient deux parties : un déplacement (FILLER) de positionnement au sommet de la pile puis la description des zones utiles.

Exemple :

Supposons qu'à un endroit du texte DML' à compiler, le niveau de la pile soit = n, et que l'ordre à compiler soit le suivant :

```
REACH R1 = ETIQ ...
```

et que la description de R1 dans le sous-shéma soit :

```
RECORD R1
  01 D-IT1 PIC 9(4)
  01 D-IT2
     02 D-IT3 PIC  X(20).
     02 D-IT4 OCCURS 10 TIMES PIC X(20).
```

Le compilateur génère dans la W.S.S. du programme :

```
01 S-STACK-i REDEFINES S-STACK.
   02 FILLER PIC X(n).
   02 ETIQ.
      03 D-IT1 PIC 9(4).
      03 D-IT2.
         04 D-IT3 PIC  X(20).
         04 D-IT4 OCCURS 10 TIMES PIC  X(20).
```

où i indique le numéro de l'ordre DML' dans le programme.
La phase de génération générera non seulement des ordres CODASYL/COBOL en procédure DIVISION mais également des définitions de zones en W.S.S. de programme généré.

Le programme généré aura donc la structure suivante :

DATA DIVISION
- W.S.S. du programme
- 01 S-STACK PIC X(n) (n est déterminé en fin de compilation)
  01 S-STACK1 REDEFINES S-STACK
  :
  :
- autres zones générées par le compilateur DML'

PROCEDURE DIVISION
Ordres COBOL d'origine.
Ordres CODASYL/COBOL équivalents aux ordres DML'.

5.3. Architecture du compilateur DML'

5.3.1. Tables et variables du compilateur

5.3.1.1. Analyse syntaxique

L'analyse syntaxique des ordres REACH, END, NEXT, EXIT, OPEN, CLOSE, SAVE implique la présence en mémoire de tables indiquant les types de données d'un sous-shéma ainsi que leurs caractéristiques.
L'ensemble des tables décrivant les sous-shémas d'un système CODASYL sont créés par le compilateur DDL'.
Lors de la génération d'un ordre OPEN de forme 1, les tables relatives au sous-shéma du programme sont préalablement amenées en mémoire.
La fig. 5.1. présente la structure de ces tables.

WITHIN-TAB : permet de retrouver la liste des areas d'une clause WITHIN.

AREA-TAB : donne la liste des areas du sous-shéma.

ITEM-TABLE: donne la liste des data-items du sous-shéma ainsi que leurs caractéristiques. Lorsqu'un data-item est un groupe de data-items, l'élément GROUPE de cette table permet de retrouver la liste des data-items dont il se compose (par la table ITEM_SUB).

REC-TAB : donne la liste des records du sous-shéma et permet de reconstituer les clauses accompagnant sa description. SET-PTR indique dans quels types de sets il participe (par la table OWNER MEM-TAB).

SET-TAB : donne la liste des sets du sous-shéma et permet de reconstituer les clauses qui le décrivent.
L'élément type indique si le member est automatic/manual et mandatory/optional.

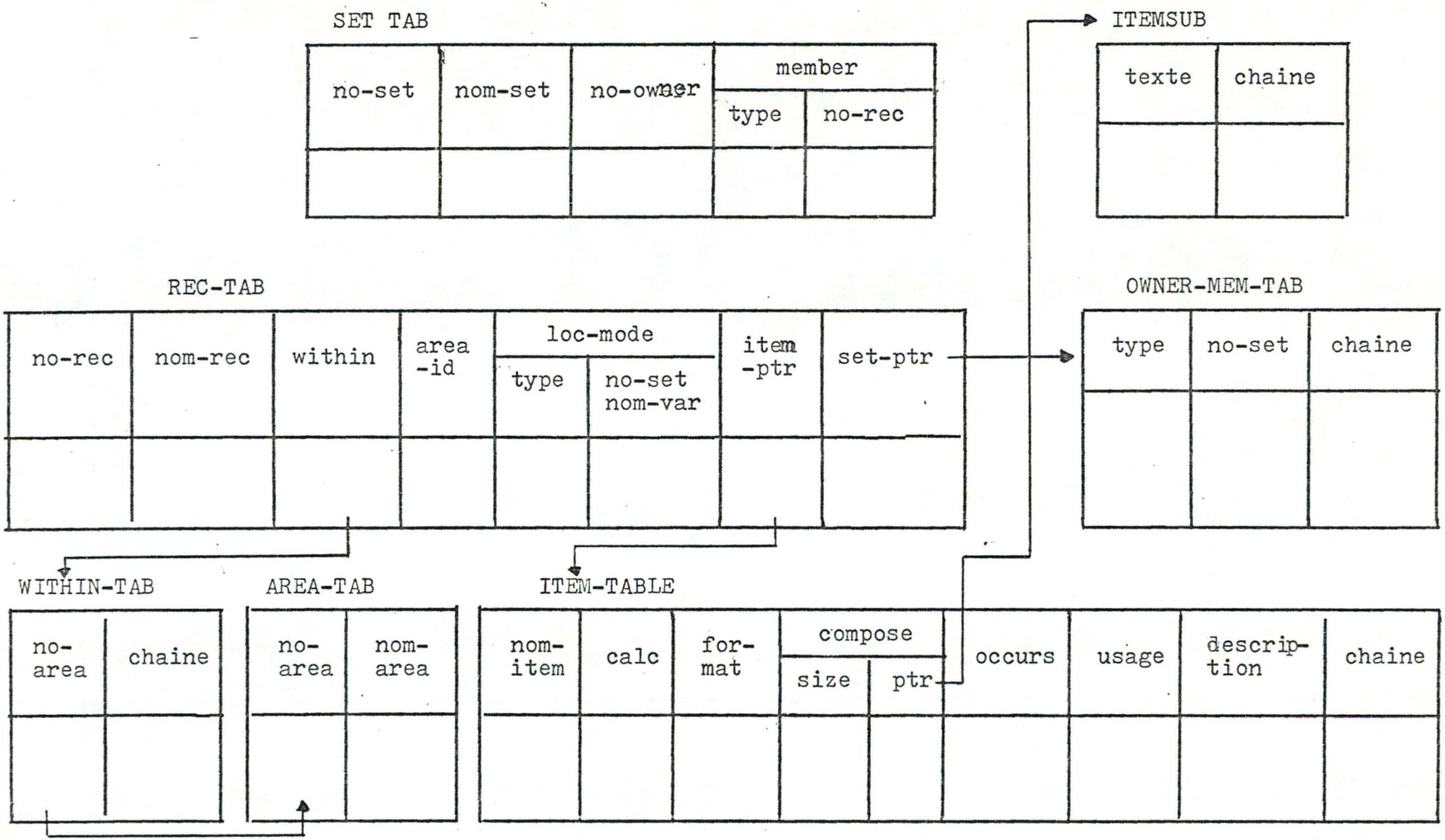

SET TAB
no-set
nom-set
no-owner
member
type
no-rec
ITEMSUB
texte
chaine
REC-TAB
no-rec
nom-rec
within
area -id
loc-mode
type
no-set nom-var
item -ptr
set-ptr
OWNER-MEM-TAB
type
no-set
chaine
WITHIN-TAB
no-area
chaine
AREA-TAB
no-area
nom-area
ITEM-TABLE
nom-item
calc
for-mat
compose
size
ptr
occurs
usage
descrip-tion
chaine

fig. 5.1.

5.3.1.2. GENERATION

Les tables et variables nécessaires à la génération des ordres DML' sont garnies après analyse lexicale, nous les présentons par la fig. 5.2.

PILE-BOUCLE : cette pile est la pile des boucles REACH et OPEN.
On lui associe un indice : top-pb.
Pour un ordre REACH elle contient :
- l'étiquette de boucle (ETIQ),
- la longueur du type de record concerné (LONG),
- le numéro dans REC-TAB du record concerné (N-RECORD),
- le type de boucle (TYPE-BOUCLE), cet élément est positionné à la génération et permettra de générer l'ordre END correspondant.
- le nombre d'areas de la clause WITHIN du type de record (NBR-A).

Pour un ordre OPEN, top-pb doit être = 1 et elle ne contient que l'étiquette de boucle.
Après analyse lexicale, top-pb contient la valeur du dernier niveau significatif de la pile; un niveau disparaît après génération du END correspondant.

LISTE-A :
pour un ordre REACH de format 1 :
cette table est constituée à partir de la clause WITHIN du type de record concerné et contient le numéro de ces areas; elle disparaît après génération de celui-ci.
pour un ordre OPEN ou CLOSE :
elle contient le numéro d'areas (ou de sets) apparaissant éventuellement dans ces ordres.

OPEN-CLOSE :
cette variable indique quel type d'ordre OPEN ou CLOSE doit être généré :

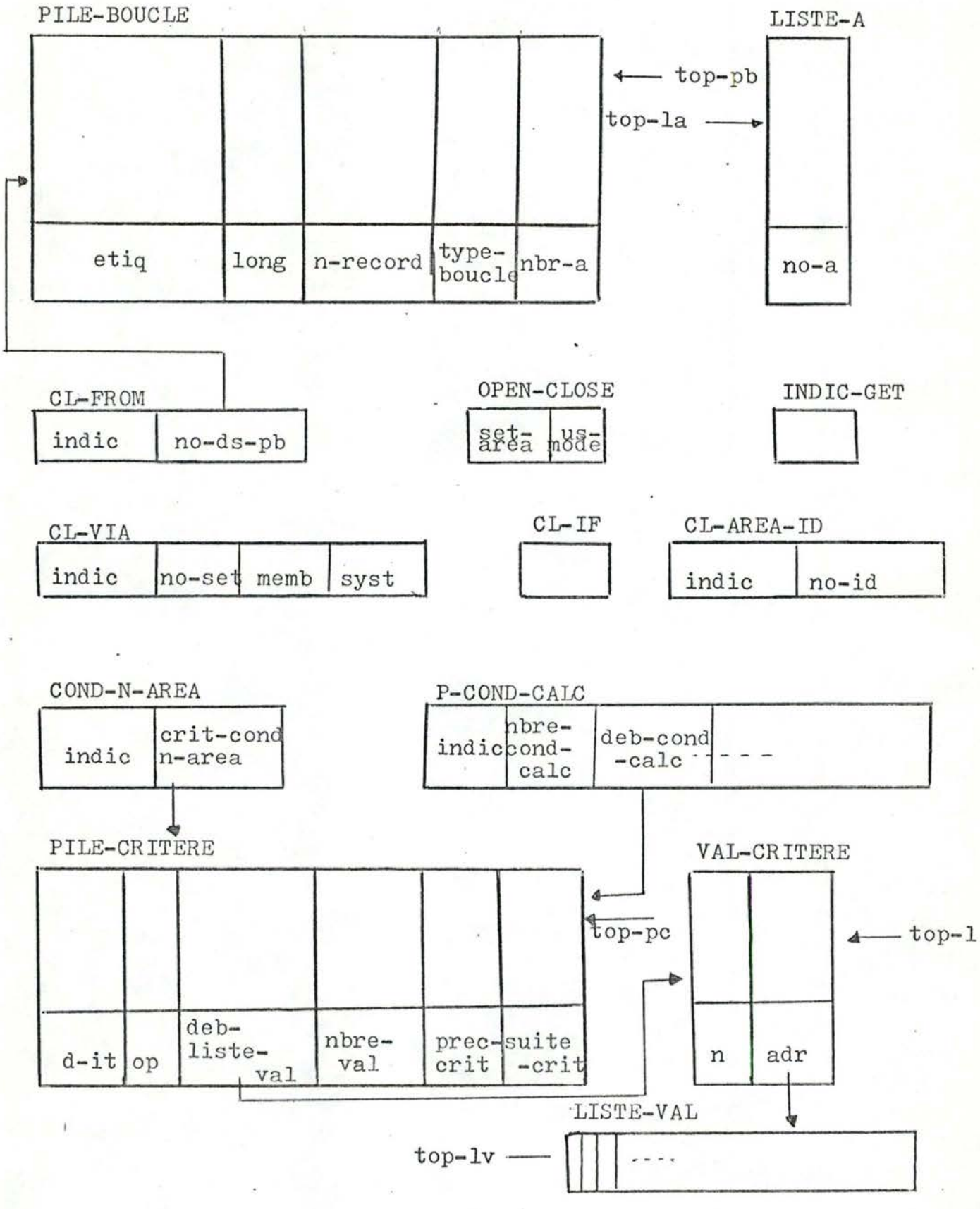

PILE-BOUCLE
LISTE-A
top-pb
top-la
etiq
long
n-record
type-boucle
nbr-a
no-a
CL-FROM
indic
no-ds-pb
OPEN-CLOSE
set-area
us-mode
INDIC-GET
CL-VIA
indic
no-set
memb
syst
CL-IF
CL-AREA-ID
indic
no-id
COND-N-AREA
indic
crit-cond n-area
P-COND-CALC
indic
nbre-cond-calc
deb-cond-calc
PILE-CRITERE
VAL-CRITERE
top-pc
top-l
d-it
op
deb-liste-val
nbre-val
prec-crit
suite-crit
n
adr
LISTE-VAL
top-lv

fig. 5.2.

si SET-AREA = 1, il s'agit d'ordres OPEN/CLOSE ALL...
= 2, il s'agit d'ordres OPEN/CLOSE ALL FOR SET...
= 3, il s'agit d'ordres OPEN/CLOSE AREA...

si US-MODE = 00,il n'y a pas d'usage mode associé à l'ordre OPEN.
= 01, l'usage mode est RETRIEVAL
= 02, l'usage mode est UPDATE
= 11, l'usage mode est PROTECTED RETRIEVAL
= 12, l'usage mode est PROTECTED UPDATE
= 21, l'usage mode est EXCLUSIVE RETRIEVAL
= 22, l'usage mode est EXCLUSIVE UPDATE.

CL-AREA-ID indique s'il existe une clause area-ID pour le type de record référencé par un ordre REACH et dans ce cas donne le numéro (NO-ID) de cette zone dans REC-TAB.

CL-FROM : cette zone indique si un ordre REACH s'accompagne d'une clause FROM ou non (INDIC) et dans ce cas le numéro de cette étiquette dans PILE-BOUCLE (NO-DS-PB).
NO-DS-PB permet aussi de localiser une étiquette relative à un NEXT, EXIT, SAVE ou END.

INDIC-GET indique si un ordre REACH s'accompagne d'un clause GET ou non.

CL-VIA : cette variable indique si un ordre REACH s'accompagne de la clause VIA et dans ce cas donne le numéro dans SET-TAB du type de set référencé. MEMB indique si le type de record de l'ordre est member (=1) ou non (=0) de ce type de set. SYST indique si l'owner est SYSTEME (=1) ou non (=0).

PILE-CRITERE :
Cette pile donne la liste des critères simples qui composent la condition d'une clause IF d'un ordre REACH. Cette pile permet d'isoler chacun de ces critères et donne pour chacun d'eux :

- le numéro du data-item (D-IT),
- l'opérateur logique (OP),
- le nombre de valeurs de ce critère (NBRE-VAL),
- l'adresse de la première valeur du critère, cette adresse est un numéro d'élément de la pile VAL-CRITERE. L'adresse des au-

tres valeurs est accessible à partir de cet élément de VAL-CRITERE.

- des pointeurs PREC-CRIT et SUITE-CRIT qui donnent respectivement l'adresse du critère précédent et suivant dans cette pile.
  PREC-CRIT = 0 pour le premier critère de la condition et
  SUITE-CRIT = 0 pour le dernier.

VAL-CRITERE :

n : donne la longueur d'une valeur d'un critère.

adc : donne l'adresse de cette valeur dans LISTE-VAL où elle est stockée caractère par caractère.

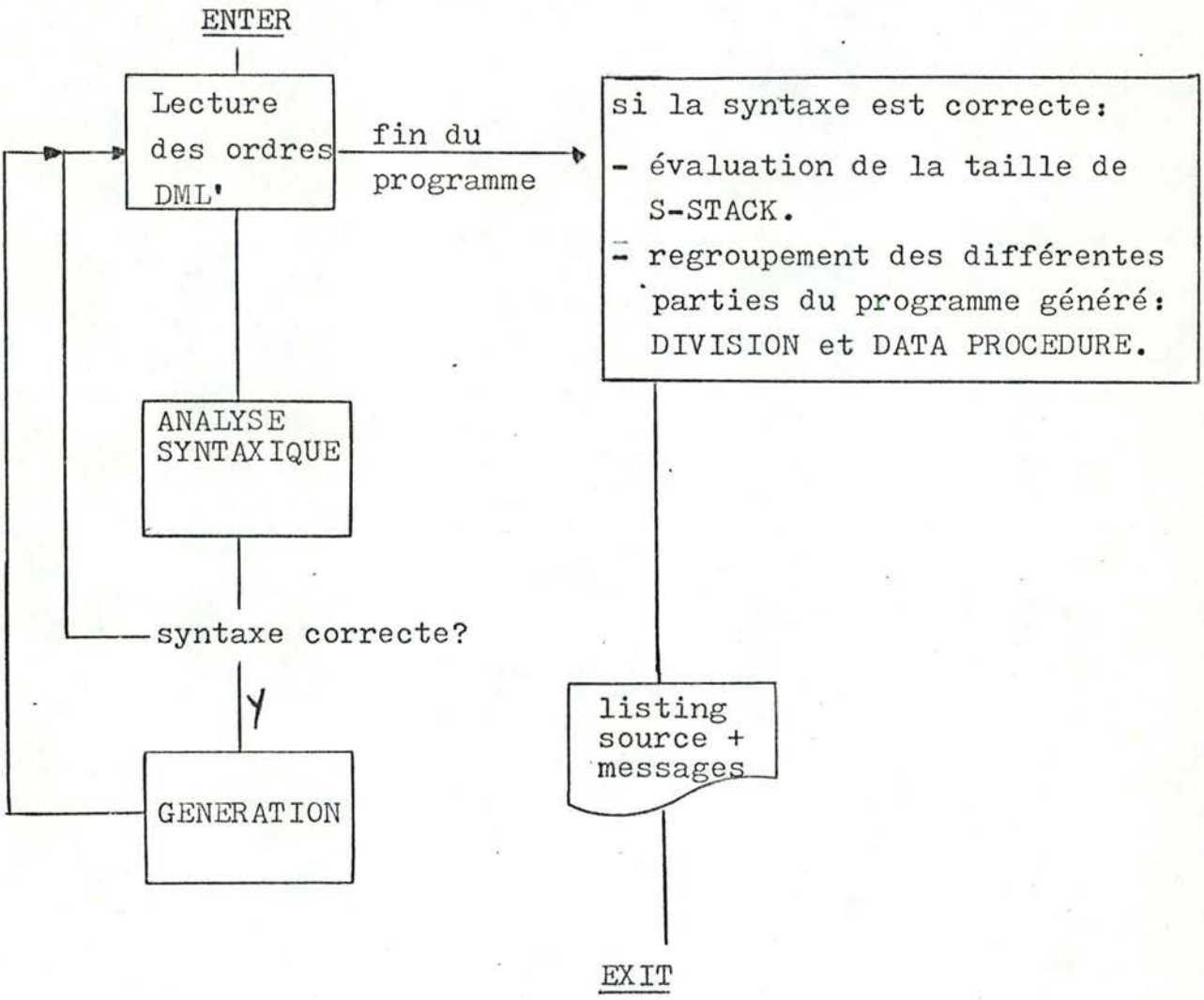
LISTE-VAL : permet de stocker les valeurs des critères de la condition.

COND-N-AREA : cette zone indique si une condition d'un ordre REACH porte sur ID-AREA et dans ce cas donne l'adresse du critère simple dans PILE-CRITERE.

P-COND-CALC. : cette pile indique si la condition d'un ordre REACH porte sur CALC, c-à-d si pour chaque CALC KEY il existe un critère simple d'égalité.
Elle donne dans ce cas le nombre de data-items de cette clause (NBRE-COND-CALC) et l'adresse des critères simples dans PILE-CRITERE (DEB-COND-CALC).

5.3.2. Les modules du compilateur.

Le module principal se présente par l'algorithme suivant :



Les programmes relatifs à l'analyse lexicale sont une suite de tests et de recherche en tables. Leur description n'offre aucun interêt, retenons toutefois qu'après cette étape de compilation les tables et variables nécéssaires à la génération sont positionnées avec les valeurs de l'ordre analysé.
Les programmes de génération se caractérisent par le texte qu'ils génèrent. Nous présentons dans la suite leurs structure ainsi que celle du texte qu'ils produisent.

5.3.2.1. REACH.

Lors de la génération d'un ordre REACH ,top-pb indique quel élément de PILE-BOUCLE caractérise cet ordre.
Toutes les variables et étiquettes de paragraphe générées pour un tel ordre sont suffixées par l'étiquette de boucle (etiq(top-pb)) afin d'individualiser les boucles de même type.
La structure générale d'une boucle générée est la suivante :

```
S-INIT-etiq( top-pb ).
      (instructions d'initialisation de boucle ).
S-BOUCLE-etiq( top-pb ).
      (sélection d'une nouvelle variable de boucle ).
S-CORPS-etiq( top-pb ).
      (instructions du corps de boucle ).
S-ITER-etiq( top-pb ).
      (repositionnement de la variable de boucle dans le
       current adéquat ).
      (branchement à S-BOUCLE-... ).
S-FIN-etiq( top-pb ).
      (instructions de sortie de boucle ).
```

Les deux derniers paragraphes correspondent à l'ordre END.
Cette structure n'est pas rigide, nous avons voulu présenter les types d'opérations à effectuer et un paragraphe portant un de ces noms en aura également la fonction; les étiquettes décrites n'apparaissent pas dans le texte généré lorsqu'il apparaît évident qu'elles ne feront pas l'objet d'ordres de branchement.

Les types de boucles se distinguent par les accès qu'elles décrivent.
Les differents types d'accès sont :

1.l'accès aux records d'un set;

2.l'accès aux records d'une ou plusieurs areas.

Ces accès peuvent être soumis à une condition et on distingue alors:

2.1.l'accès aux records d'une au plusieurs areas sans condition;

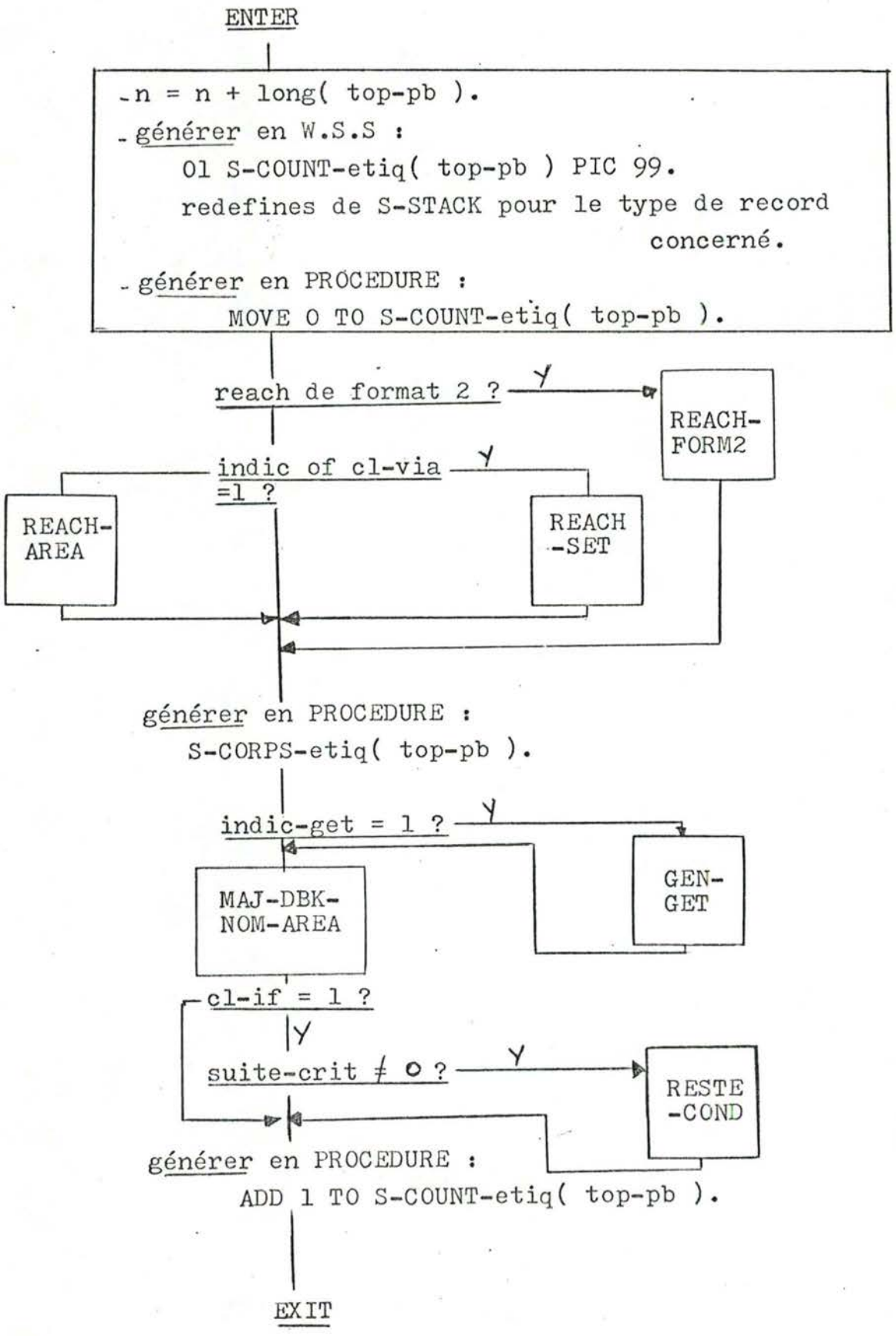
2.2.l'accès aux records d'une ou plusieurs areas avec une condition portant sur CALC et évevtuellement sur le data-item ID-AREA;

2.3.l'accès aux records d'une ou plusieurs areas avec une condition portant sur le data-item ID-AREA.

Le DBMS-20 ne reconnaît pas la notion de SEARCH KEY, c'est pourquoi nous ne distinguons pas l'accès aux records d'un set sans condition du cas où il y en aurait une.

Ces differents types d'accès déterminent les ordres Codasyl générés pour l'initialisation et l'itération (S-BOUCLE-...).

Le programme relatif à la génération de l'ordre REACH s'articule selon l'algorithme suivant :



MAJ-DBK-NOM-AREA.

Ce bloc génère en procedure :

```
MOVE CURRENCY STATUS FOR RUN-UNIT TO D-B-K OF etiq( top-pb ).
MOVE AREA-NAME TO ID-AREA OF etiq( top-pb ).
```

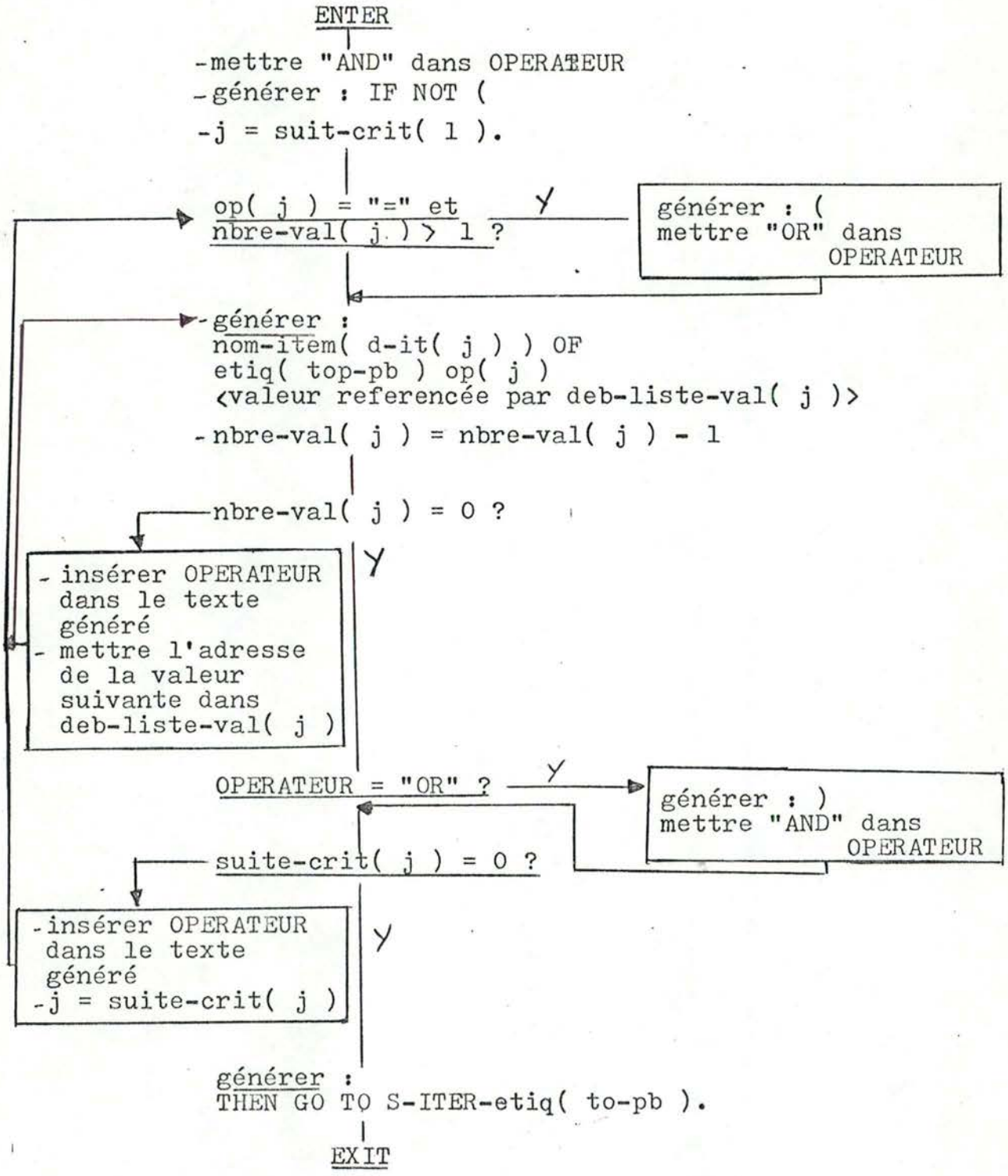
GEN-GET.

Ce bloc génère en procedure :

```
GET nom-rec( n-record( top-pb ) ).
MOVE CORRESPONDING nom-rec( n-record( top-pb ) ) TO etiq( top-pb ).
IF ERROR-STATUS = 0510 MOVE 0302 TO S-ERROR.
IF ERROR-STATUS = 0519 MOVE 0301 TO S-ERROR.
IF ERROR-STATUS = 0552 MOVE 0303 TO S-ERROR.
IF ERROR-STATUS = 0553 MOVE 0304 TO S-ERROR.
IF ERROR-STATUS = 0554 MOVE 0305 TO S-ERROR.
```

RESTE-COND.

Ce bloc transforme toute ou partie de la condition d'une clause IF sous forme d'un ordre Cobol.



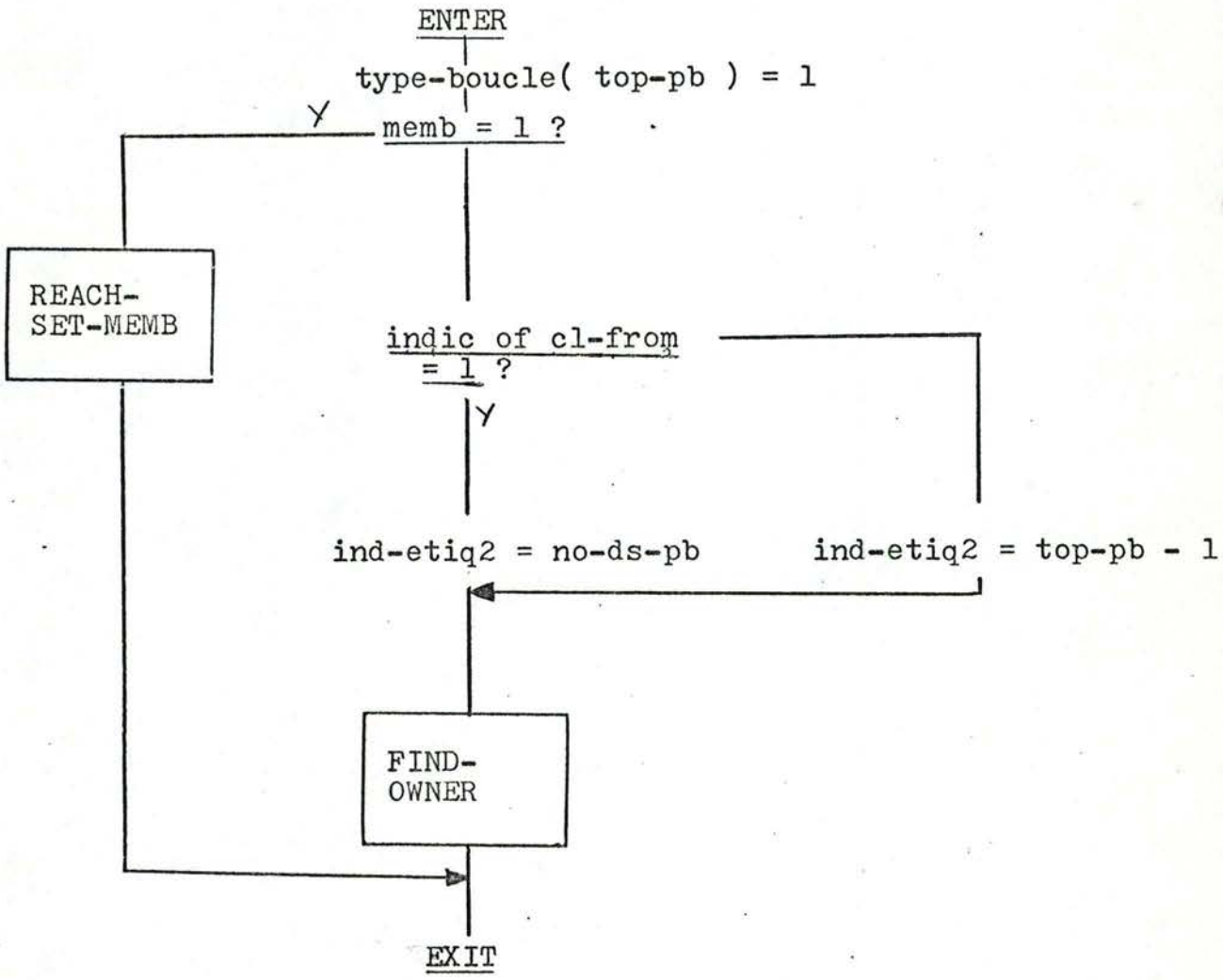
REACH-FORM2.

Ce bloc génère en procedure :

```
FIND nom-rec( n-record( top-pb ) ) USING D-B-K OF etiq( top-pb ).
IF ERROR-STATUS = 0301 MOVE 0213 TO S-ERROR
                       GO TO S-FIN-etiq( top-pb ).
IF ERROR-STATUS = 0306 MOVE 0212 TO S-ERROR
                       GOT TO S-FIN-etiq( top-pb ).
IF ERROR-STATUS = 0326 MOVE 0212 TO S-ERROR
                       GO TO S-FIN-etiq( top-pb ).
IF ERROR-STATUS = 0310 MOVE 0205 TO S-ERROR
                       GO TO S-FIN-etiq( top- pb ).
```

REACH-SET.

Ce bloc génère en procedure l'équivalent d'un ordre REACH avec clause VIA. Le DBMS-20 ignore la notion de SEARCH KEY, c'est pourquoi nous n'envisageons pas le cas d'accés direct pour un set.



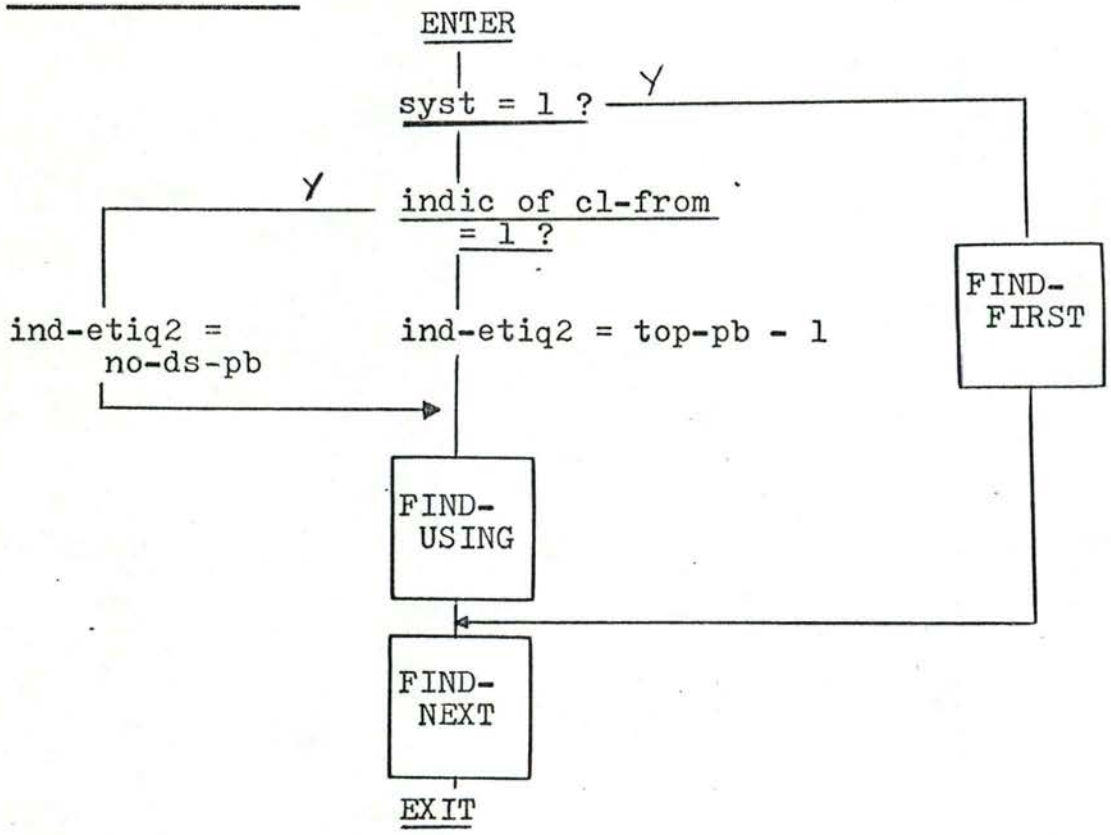
FIND-OWNER.

Ce bloc génère en procedure :

```
FIND nom-rec( n-record( ind-etiq2 ) ) USING D-B-K OF etiq( ind-etiq2 )
IF ERROR-STATUS = 0301 MOVE 0204 TO S-ERROR GO TO S-FIN-etiq( top-pb )
IF ERROR-STATUS = 0310 MOVE 0205 TO S-ERROR GO TO S-FIN-etiq( top-pb )
IF ERROR-STATUS = 0326 MOVE 0202 TO S-ERROR GO TO S-FIN-etiq( top-pb )
```

```
IF ERROR-STATUS = 0306 MOVE 0202 TO S-ERROR GO TO S-FIN-etiq( top-pb )
FIND OWNER RECORD OF nom-set( no-set of cl-via ) SET.
IF ERROR-STATUS = 0301 MOVE 0206 TO S-ERROR GO TO S-FIN-etiq( top-pb )
IF ERROR-STATUS = 0318 MOVE 0207 TO S-ERROR GO TO S-FIN-etiq( top-pb )
IF ERROR-STATUS = 0322 MOVE 0203 TO S-ERROR GO TO S-FIN-etiq( top-pb )
```

REACH-SET-MEMB.



FIND-USING.

Lès ordres Codasyl/Cobol générés sont identiques à l'ordre FIND...USING et les ordres de test de valeurs de ERROR-STATUS relatifs à cet ordre générés dans le bloc FIND-OWNER.

FIND-FIRST.

Ce bloc génère en procedure :

```
FIND FIRST nom-rec( n-record( top-pb ) ) RECORD OF
                              nom-set( no-set of cl-via ) SET.
IF ERROR-STATUS = 0301 MOVE 0206 TO S-ERROR GO TO S-FIN-etiq( top-pb ).
IF ERROR-STATUS = 0318 MOVE 0207 TO S-ERROR GO TO S-FIN-etiq( top-pb ).
```

```
IF ERROR-STATUS = 0310 MOVE 0205 TO S-ERROR GO TO S-FIN-etiq( top-pb ).
IF ERROR-STATUS = 0322 MOVE 0203 TO S-ERROR GO TO S-FIN-etiq( top-pb ).
GO TO S-CORPS-etiq( top-pb ).
```

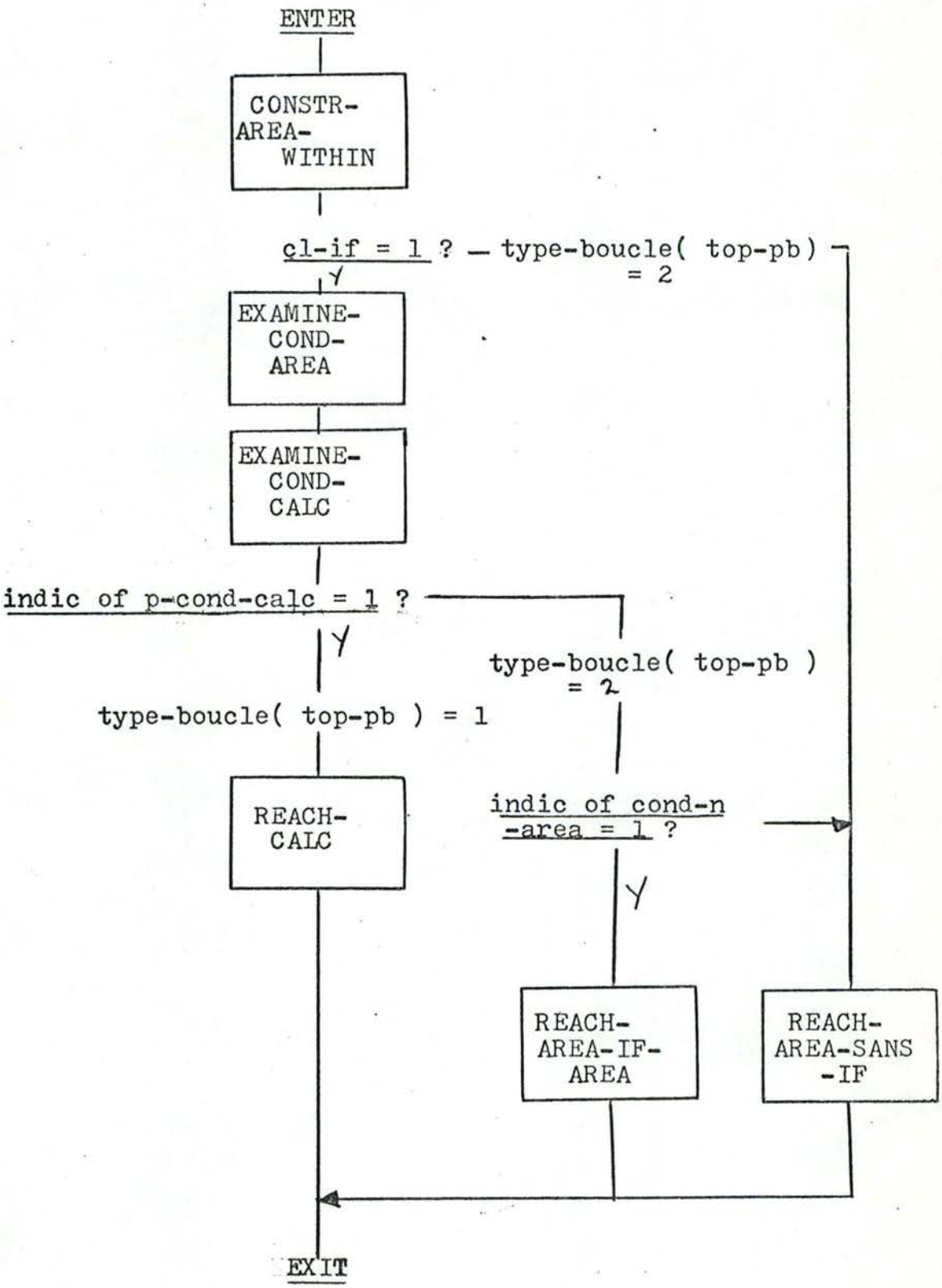
FIND-NEXT.

Ce bloc génère en procedure :

```
S-BOUCLE-etiq( top-pb ).
      FIND NEXT nom-rec( n-record( top-pb ) ) RECORD OF
                          nom-set( no-set of cl-via ) SET.
      IF ERROR-STATUS = 0301 MOVE 0206 TO S-ERROR
                             GO TO S-FIN-etiq( top-pb ).
      IF ERROR-STATUS = 0318 MOVE 0207 TO S-ERROR
                             GO TO S-FIN-etiq( top-pb ).
      IF ERROR-STATUS = 0307 GO TO S-FIN-etiq( top-pb ).
      IF ERROR-STATUS = 0326 MOVE 0201 TO S-ERROR
                             GO TO S-FIN-etiq( top-pb ).
```

REACH-AREA.

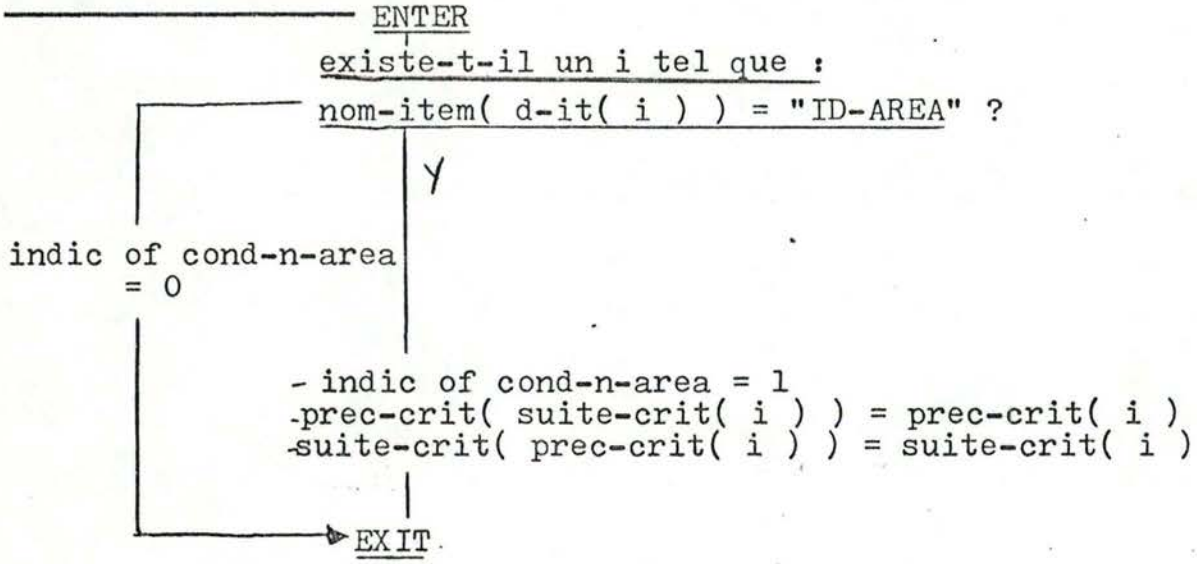
Ce bloc génère l'équivalent d'un ordre REACH sans VIA.



CONSTR-AREA-WITHIN.

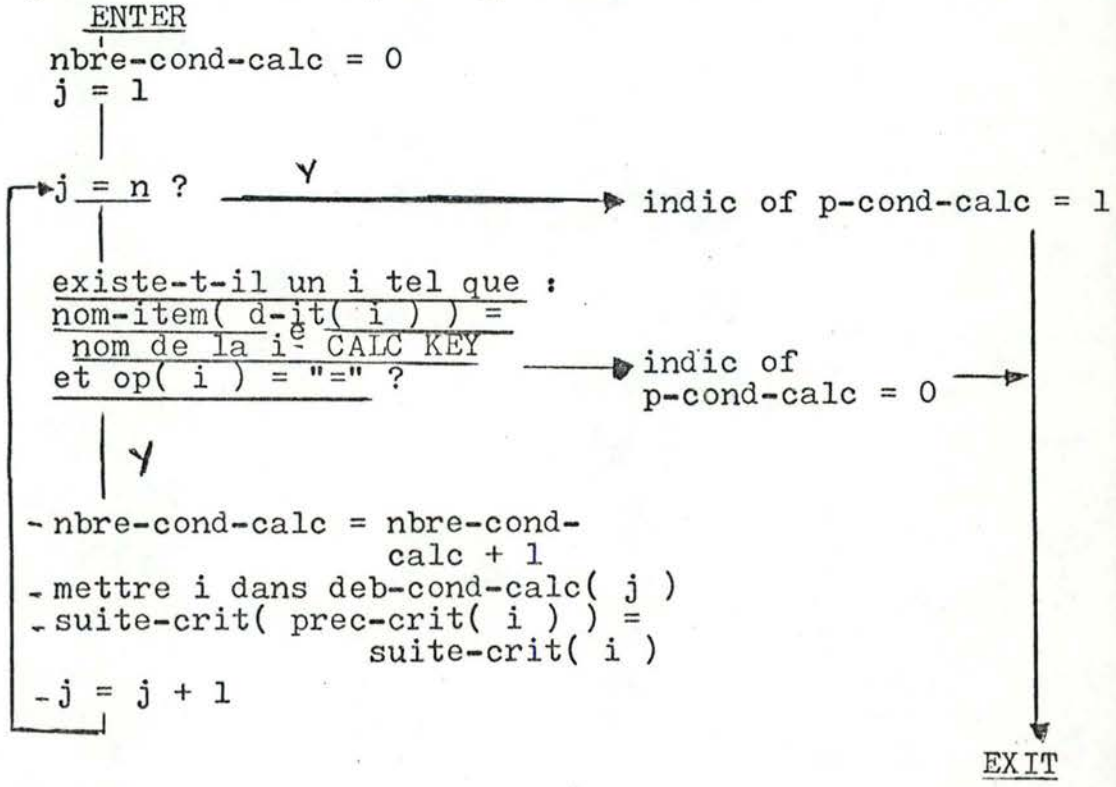
Ce bloc construit LISTE-A avec les areas de la clause WITHIN du type de record référencé par le sommet(top-pb) de PILE-BOUCLE.

EXAMINE-COND-AREA.



EXAMINE-COND-CALC.

Ce bloc examine une condition afin de vérifier si elle porte sur CALC. Supposons qu'il y ait n CALC KEYS, l'algorithme est alors :



REACH-AREA-SANS-IF.

Ce bloc produit un texte équivalent à un ordre REACH sans clause VIA et avec une condition éventuelle mais ne permettant aucun accès direct.
Il génère : en W.S.S :

- 01 S-K-etiq(top-pb) pic 99.

en PROCEDURE :

- MOVE 0 TO S-K-etiq(top-pb).

-pour chaque area de la clause within, c-a-d pour j variant de 1 à nbr-a(top-pb) :

```
    S-INIT-j-etiq( top-pb ).
      ADD 1 TO S-K-etiq( top-pb ).
       FIND FIRST nom-rec( n-record( top-pb ) ) RECORD OF
       nom-area( liste-a( j ) ) AREA.
       IF ERROR-STATUS = 0301 MOVE 0209 TO S-ERROR
                              GO TO S-INIT-j+1-etiq( top-pb ).
       IF ERROR-STATUS = 0318 MOVE 0210 TO S-ERROR
                              GO TO S-INIT-j+1-etiq( top-pb ).
       IF ERROR-STATUS = 0310 MOVE 0205 TO S-ERROR
                              GO TO S-FIN-etiq( top-pb ).
       IF ERROR-STATUS = 0318 GO TO S-INIT-j+1-etiq( top-pb ).
       GO TO S-CORPS-etiq( top-pb ).
    S-BOUCLE-j-etiq( top-pb ).
       FIND NEXT nom-rec( n-record( top-pb ) ) RECORD OF
      nom-area( liste-area( j ) ) AREA.
       IF ERROR-STATUS = 0301 MOVE 0209 TO S-ERROR
                              GO TO S-INIT-j+1-etiq( top-pb ).
       IF ERROR-STATUS = 0306 MOVE 0211 TO S-ERROR
                              GO TO S-INIT-j+1-etiq( top-pb ).
      IF ERROR-STATUS = 0307 GO TO S-INIT-j+1-etiq( top-pb ).
       GO TO S-CORPS-etiq( top-pb ).
```

-pour j = nbr-a(top-pb) + 1 :

```
    S-INIT-j-etiq( top-pb ).
       GO TO S-FIN-etiq( top-pb ).
```

REACH-AREA-IF-AREA.

Ce bloc produit un texte équivalent à un ordre REACH sans VIA et avec une condition portant sur ID-AREA.
Il génère : en W.S.S. :

```
01 S-VAL-AREAC-etiq( top-pb ) OCCURS nbre-val( crit-cond-n-area )
                                TIMES PIC X(30).
```

*ce tableau est destiné à contenir les valeurs du critère simple de la condition portant sur ID-AREA.

```
01 S-K-etiq( top-pb ) PIC 99.
```

en PROCEDURE :

```
-     MOVE 0 TO S-K-etiq( top-pb ).
```

- pour chaque valeur du critère portant sur ID-AREA, c-a-d : pour i variant de 1 à nbre-val(crit-cond-n-area) :

```
     MOVE ⟨ i^eme valeur du critère ⟩ TO
                        S-VAL-AREAC-etiq( top-pb ).
```

- pour chaque area de la clause WITHIN, c-a-d pour j variant de 1 à nbr-a(top-pb) :
  - S-INIT-j-etiq(top-pb).

```
     SET S-I TO 1.
     ADD 1 TO S-K-etiq( top-pb ).
```

  - si op(crit-cond-n-area) = "=" :

```
    SEARCH S-VAL-AREAC-etiq( top-pb ) VARYING S-I
    AT END GO TO S-INIT -j+1-etiq( top-pb )
    WHEN S-VAL-AREAC-etiq( top-pb )( S-I ) = "nom-area( liste-a(j) )"
    NEXT SENTENCE.
```

  - sinon :

```
    SEARCH S-VAL-AREAC-etiq( top-pb ) VARYING S-I
    AT END NEXT SENTENCE
    WHEN S-VAL-AREAC-etiq( top-pb )( S-I ) = "nom-area( liste-a(j) )"
    GO TO S-INIT-j+1-etiq( top-pb ).
```

  - ordre FIND ...FIRST .... comme par REACH-AREA-SANS-IF
  - S-BOUCLE- ... : comme pour REACH-AREA-SANS-IF
- pour j = nbr-a(top-pb) + 1 : comme pour REACH-AREA-SANS-IF

REM : le texte généré est équivalent à celui du cas précédent; l'ordre SEARCH qui est généré en plus sert à vérifier si une area présente dans la clause WITHIN est acceptable par la condition.

REACH-CALC.

Ce bloc génère l'équivalent d'un ordre REACH sans VIA et avec une condition portant sur CALC et éventuellement sur ID-AREA. Nous ne faisons pas un cas particulier de cette dernière condition car les ordres d'accès à générer sont identiques dans les deux cas, seule la liste des areas concernées peut différer.
Un accès basé sur des CALC KEYS se sert des clauses WITHIN et AREA-ID pour l'identification d'une area, c'est pourquoi les valeurs possibles pour les area concernées par l'ordre seront gérées par le programme; ceci évite de devoir générer les ordres d'accès aussi souvent qu'il existe d'areas possibles (comme il a fallut le faire pour les deux cas précédents).
Il génère en W.S.S :

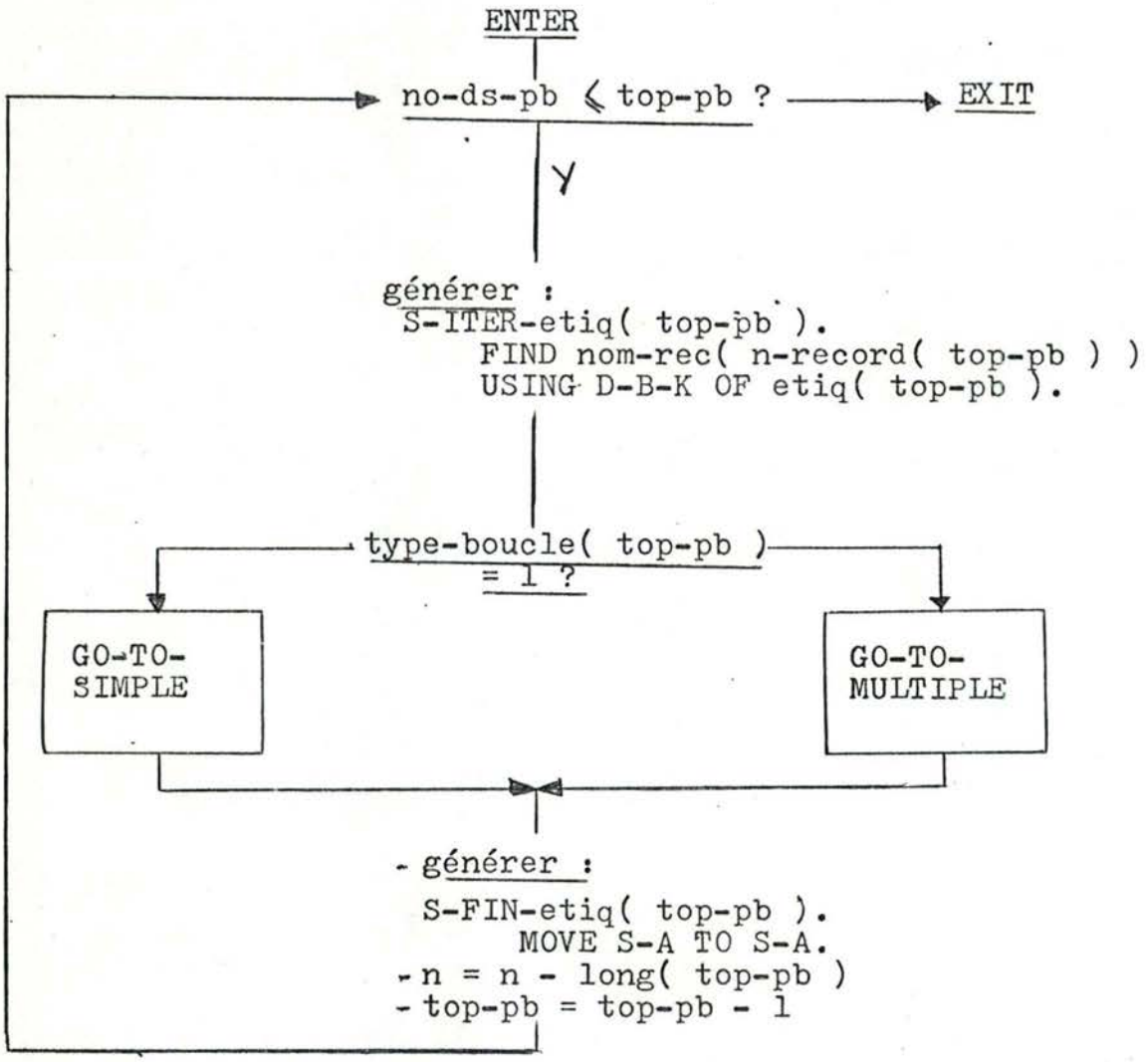
- si il existe un critère portant sur ID-AREA :
  01 S-VAL-AREAC-etiq(top-pb) OCCURS nbre-val(crit-cond-n-area) TIMES PIC X(30).
- un tableau destiné à contenir les noms des areas de la clause within :
  01 S-VAL-AREA-etiq(top-pb) OCCURS nbr-a(top-pb) TIMES PIC X(30).
  un indice pour ce tableau :
  01 S-IND-A-etiq(top-pb) PIC 99.
- pour chaque CALC KEY : un tableau destiné à contenir les valeurs de la condition, c-a-d pour j variant de 1 à nbre-cond-calc :
  01 S-VAL-CALC-j-etiq(top-pb) OCCURS nbre-val(deb-cond-calc(j)) TIMES.
  |
  |
  < description de ce data-item, conformement au sous-shéma >
  |
  |
- des indices pour ces tableaux:
  01 S-IND-C-etiq(top-pb) OCCURS nbre-cond-calc TIMES PIC 99.
  * un élément de ce tableau ,soit S-IND-C-etiq(top-pb)(L), est l'indice associé à un tableau S-VAL-CALC- ... ,soit S-VAL-CALC-L-etiq(top-pb).
- 01 S-NBRE-FOIS-etiq(top-pb) PIC 9.

Il génère en procedure :

- le passage des valeurs de la condition :
  - pour CALC :
    pour j variant de 1 à nbre-cond-calc :
    MOVE 1 TO S-IND-C-etiq(top-pb)(j).
    pour i variant de 1 à nbre-val(deb-cond-calc(j)) :
    MOVE ⟨ $i^{ème}$ valeur du critère portant sur la $j^{ème}$ CALC KEY ⟩ TO S-VAL-CALC-j-etiq(top-pb)(i).
  - pour ID-AREA éventuellement :
    pour j variant de 1 à nbre-val(crit-cond-n-area) :
    MOVE ⟨ $j^{ème}$ valeur du critère ⟩ TO
    S-VAL-AREAC-etiq(top-pb)(j).
- le passage des valeurs de la clause WITHIN :
  -pour j variant de 1 à nbr-a(top-pb) :
  MOVE "nom-area(liste-a(j))" TO S-VAL-AREA-etiq(top-pb)
  - MOVE 1 TO S-IND-A-etiq(top-pb).
- si il existe un critère portant sur ID-AREA :
  S-CON-BOUCLE-etiq(top-pb).
  SET S-I TO 1.
  si op(crit-cond-n-area) = "=" :
  SEARCH S-VAL-AREAC-etiq(top-pb) VARYING S-I
  AT END GO TO S-NOUVEL-ITER-etiq(top-pb)
  WHEN S-VAL-AREA-etiq(top-pb)(S-IND-A-etiq(top-pb))
  = S-VAL-AREAC-etiq(top-pb)(S-I) NEXT SENTENCE.
  sinon :
  SEARCH S-VAL-AREAC-etiq(top-pb) VARYING S-I
  AT END NEXT SENTENCE WHEN S-VAL-AREAC-etiq(top-pb)(S-I)
  = S-VAL-AREA-etiq(top-pb)(S-IND-A-etiq(top-pb)
  GO TO S-NOUVEL-ITER-etiq(top-pb).
  - S-PRA-BOUCLE-etiq(top-pb).
  MOVE 0 TO S-NBRE-FOIS-etiq(top-pb).
  S-BOUCLE-etiq(top-pb).
  - pour j variant de 1 à nbre-cond-calc :
  MOVE S-VAL-CALC-j-etiq(top-pb)(S-IND-C-etiq(top-pb))
  TO nom-item(d-it(deb-cond-calc(j)) OF
  nom-rec(n-record(top-pb)).
  - si il existe une clause AREA-ID :
  MOVE S-VAL-AREA-etiq(top-pb)(S-IND-A-etiq(top-pb))
  TO area-id(no-id).

```
      IF S-NBRE-FOIS-etiq( top-pb ) = 1 GO TO S-ITER-etiq( top-pb )
      MOVE 1 TO S-NBRE-FOIS-etiq( top-pb ).
      FIND FIRST nom-rec( n-record( top-pb ) ) RECORD.
      IF ERROR-STATUS = 0326 GO TO S-NOUVEL-ITER-etiq( top-pb ).
      IF ERROR-STATUS = 0301 MOVE 0209 to S-ERROR
                             GO TO S-NOUVEL-ITER-etiq( top-pb ).
      IF ERROR-STATUS = 0318 MOVE 0210 TO S-ERROR
                             GO TO S-NOUVEL-ITER-etiq( top-pb ).
      IF ERROR-STATUS = 0310 MOVE 0205 TO S-ERROR
                             GO TO S-FIN-etiq( top-pb ).
      GO TO S-CORPS-etiq( top-pb ).
   S-ITER-etiq( top-pb ).
      FIND NEXT DUPLICATES WITHIN nom-rec( n-record( top-pb ) )
      RECORD.
      IF ERROR-STATUS = 0326 GO TO S-NOUVEL-ITER-etiq( top-pb ).
      IF ERROR-STATUS = 0301 MOVE 0209 TO S-ERROR
                             GO TO S-NOUVEL-ITER-etiq( top-pb ).
   S-NOUVEL-ITER-etiq( top-pb ).
      ADD 1 TO S-IND-A-etiq( top-pb ).
      IF S-IND-A-etiq( top-pb ) NOT > nbr-a( top-pb )
      si il existe une condition portant sur ID-AREA :
                             GO TO S-PRA-BOUCLE-etiq( top-pb ).
      sinon :                GO TO S-CON-BOUCLE-etiq( top-pb ) .
      MOVE 0 TO S-IND-A-etiq( top-pb ).
- pour j variant de 1 à nbre-cond-calc :
      ADD 1 TO S-IND-C-etiq( top-pb )( j ).
      IF S-IND-C-etiq( top-pb )( j ) NOT GT
      nbre-val( deb-cond-calc( j ) ) GO TO S-NOUVEL-ITER-etiq(
                                                          top-pb ).
      si j = nbre-cond-calc : GO TO S-FIN-etiq( top-pb ).
      sinon :
      MOVE 1 TO S-IND-C-etiq( top-pb )( j ).
```

5.3.2.2. END.



L'instruction générée après l'etiquette de fin de boucle évite à la procédure de comporter un paragraphe éventuellement vide (la suite du texte comporte les instructions suivantes du texte source).

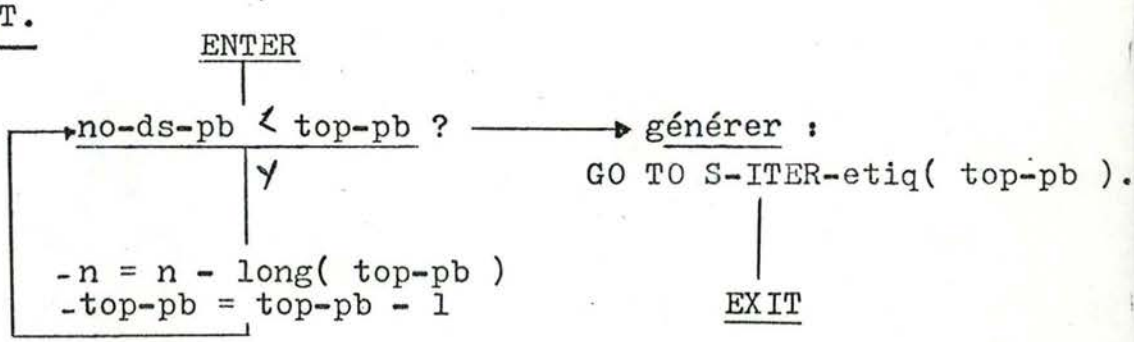
GO-TO-SIMPLE. Ce bloc génère en procedure :

```
IF ERROR-STATUS = 0301 MOVE 0201 TO S-ERROR GO TO S-FIN-etiq( top-pb )
IF ERROR-STATUS = 0326 MOVE 0201 TO S-ERROR GO TO S-FIN-etiq( top-pb )
IF ERROR-STATUS = 0306 MOVE 0201 TO S-ERROR GO TO S-FIN-etiq( top-pb )
GO TO S-BOUCLE-etiq( top-pb ).
```

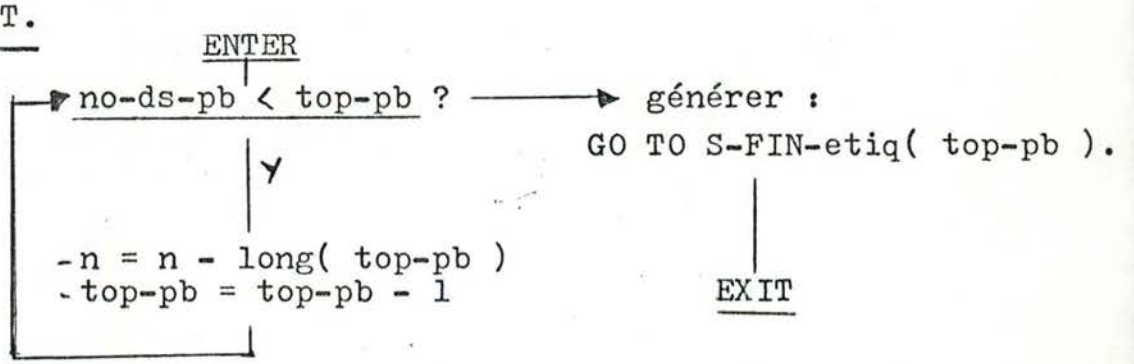
GO-TO-MULTIPLE. Ce bloc génère en procedure :

```
IF ERROR-STATUS = 0301 MOVE 0209 TO S-ERROR GO TO
⟨pour i variant de 2 à nbr-a( top-pb ) : insérer dans le texte généré :
                                          S-INIT-i-etiq( top-pb )⟩
DEPENDING ON S-K-etiq( top-pb ).
IF ERROR-STATUS = 0306 MOVE 0211 TO S-ERROR ...
      ⟨même branchement conditionnel que ci-dessus⟩.
IF ERROR-STATUS = 0326 MOVE 0211 TO S-ERROR ...
      ⟨même branchement conditionnel que ci-dessus⟩.
GO TO
⟨pour i variant de 1 à nbr-a( top-pb ) : insérer dans le texte généré :
                                          S-INIT-i-etiq( top-pb )⟩
DEPENDING ON S-K-etiq( top-pb ).
```

5.3.2.3. NEXT.



5.3.2.4. EXIT.



5.3.2.5. OPEN.

Forme 1.

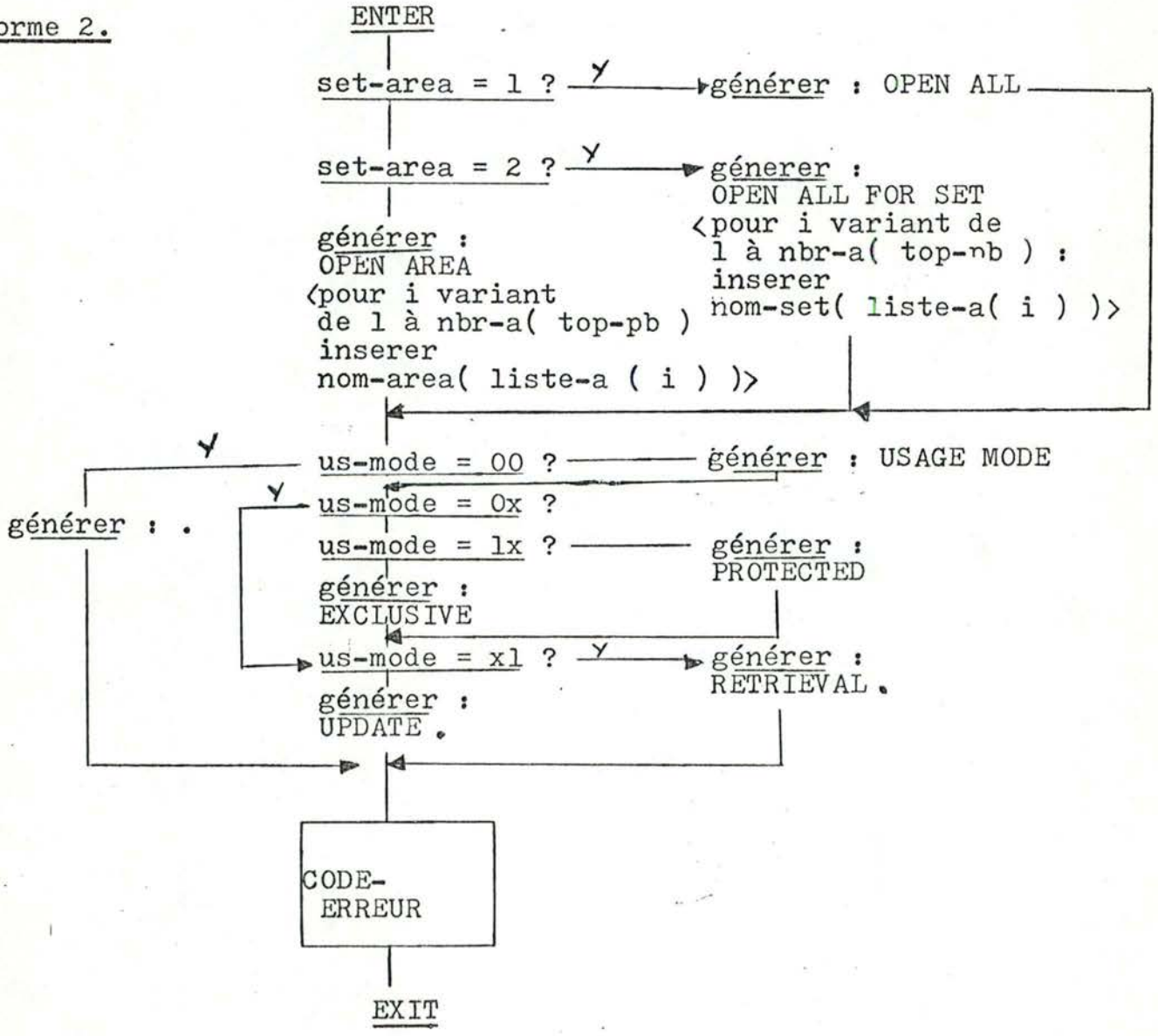
Cet ordre se présente sous la forme :

OPEN nom de sous-shéma = étiq SUB-SHEMA OF
nom de shéma SHEMA.

La génération consiste à :

- amener les tables de l'analyse syntaxique en mémoire;
- placer étiq dans etiq(1);
- générer en procedure : INVOKE nom de sous-shéma SUB-SHEMA OF nom de shéma SHEMA.

Forme 2.



CODE-ERREUR.

générer en procedure :

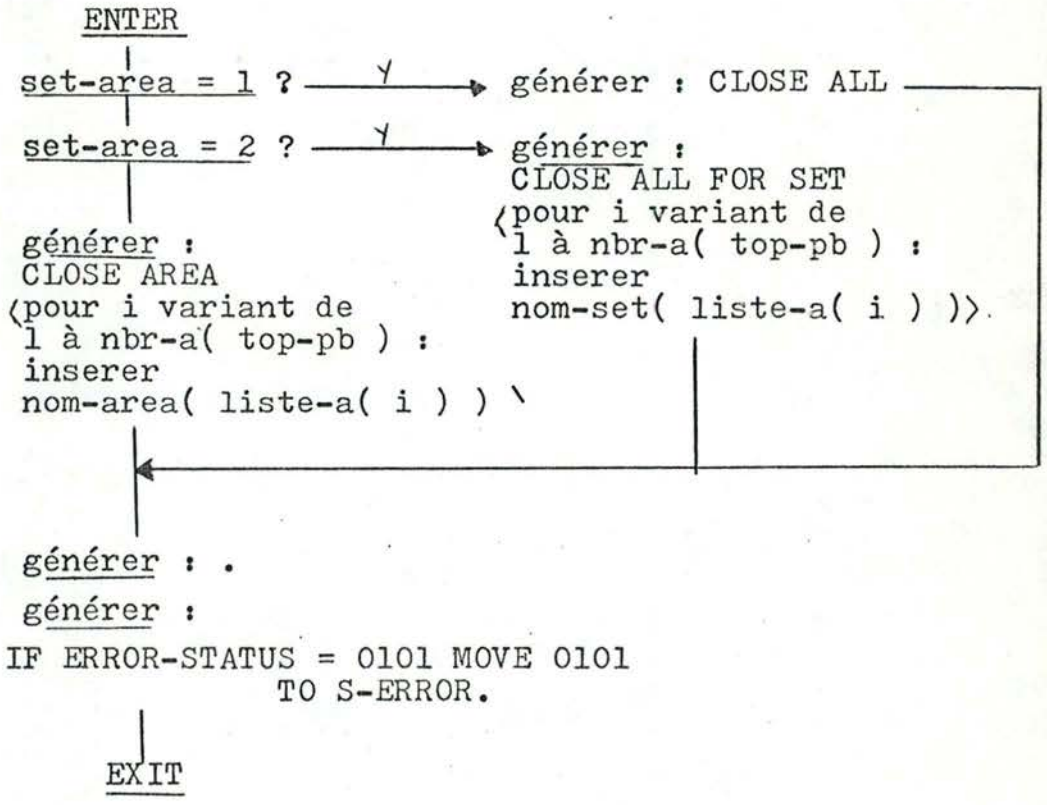
```
IF ERROR-STATUS = 0910 MOVE 0001 TO S-ERROR.
IF ERROR-STATUS = 0928 MOVE 0002 TO S-ERROR.
IF ERROR-STATUS = 0929 MOVE 0003 TO S-ERROR.
```

5.3.2.6. CLOSE.

Forme 1.

Générer en procedure : S-FIN-etiq(1).
CLOSE ALL.

Forme 2.



5.3.2.7. SAVE.

Cet ordre prend la forme :

@ SAVE ⟨nom etiquette⟩ TO ⟨var. travail⟩ .

L'ordre généré est :

MOVE D-B-K OF ⟨ nom etiquette⟩ TO ⟨var. travail⟩ .

Chapitre six : Synthèse et conclusions.

Les langages que nous avons décrit dans ce travail ont pour objectif de simplifier le travail de l'utilisateur.
Cette simplification est dûe d'une part à l'occultation de certains mécanismes délicats tels que le LOCATION MODE, la SET OCCURENCE SELECTION et les CURRENTS et d'autre part à l'uniformisation d'autres concepts regroupés sous les notions d'IDENTIFIER, d'ACCESS KEY et d'ID-AREA.
Le modèle CODASYL ainsi simplifié permet l'exploitation aisée d'une base de données et la notion de structure de bloc introduite dans le DML' entraîne l'écriture de programmes clairs et bien structurés.
Ces facilités risquent parfois d'aller à l'encontre d'un autre critère non moins important : l'efficacité à l'exploitation :

- les opérations générées par le compilateur DML' sont dans certains cas redondantes et volumineuses, exemple : la gestion des currents et la prise en charge de la SOS;
- certaines facilités offertes par le système CODASYL ne sont pas pleinement exploitées, ce sont notamment les possibilités offertes par le LOCATION MODE DIRECT (qui permet une optimisation physique en rangeant un record à un endroit voulu) et la SET OCCURENCE SELECTION.

Ces inconvénients ne sont toutefois pas évidents car ces mécanismes, offrant des avantages certains, n'en sont pas moins complexes et exigent une connaissance parfaite de leur utilisation; leur prise en charge automatique par le compilateur DML' évite de nombreux risques d'erreurs au programmeur non averti.

A partir de ce travail, de nombreux prolongements sont possibles. L'implémentation réalisée limite les applications aux programmes de consultation et une première extension consiste à permettre également des applications de mise à jour.
D'autre part il serait avantageux d'optimiser le compilateur : aucun cas particulier n'a été envisagé et certains d'entre eux permettent des simplifications interessantes à la génération. Ces simplifications ne doivent toutefois être envisagées que dans la mesure où elle n'alourdissent pas le compilateur.
Il serait également interessant de confronter les nouveaux langages avec différents utilisateurs afin de tester les avantages des options prises pour la définitions de certains ordres, exemple : les ordres OPEN et CLOSE d'areas. Ceci permettrait également de définir précisement les utilisateurs concernés et d'apporter des modifications éventuelles afin de mieux répondre à leurs besoins.
Enfin, nous croyons que ce travail peut rapidement être réalisé pour d'autres SGBD de type CODASYL : les langages étant définis sur base des normes CODASYL, la transposition dans un système existant doit être aisée.

Références et bibliographie

1. - DEHENEFFE, HAINAUT, HENNEBERT, LECHARLIER, PAULUS
"Système de conception et d'exploitation d'une base de données".
Première partie : Modèles et Langages
Deuxième partie : Manuel de Référence des Langages
Quatrième partie: Le SYSTEME SPHINX
Utilisation, fonctionnement et description interne.
Projet de recherche C.I.P.S. - rapports finaux - Publications de l'institut d'informatique de Namur - 1978.

2. - DELVAUX
"Implémentation d'un langage de haut niveau de manipulation de bases de données".
Mémoire de fin d'étude - 1977

3. - LALOUX
"Implémentation du modèle d'accès par lui-même".
Mémoire de fin d'étude - 1976

4. - LECHARLIER, HAINAUT
"Modèles, langages et système pour la conception et l'exploitation de bases de données".
Publications de l'institut d'informatique de Namur - 1979

5. - CODASYL : "Data base task group 1971 Report", ACM, 1971.

6. - HAINAUT
"Etude de la Set Occurence Sélection dans les rapports Codasyl".
Publications de l'institut d'informatique à Namur - 1979

7. - DATE
"An introduction to data base systems" second edition.
ADDISON-WESLEY - 1977

8. - FRY, SIBLEY
"Evolution of data-base management systems".
ACM Computing Survey - volume 8, number 2 march 1976

9. - TAYLOR, FRANK
"Codasyl data-base management systems"
ACM Computing Survey - volume 8 , number 1 march 1976

10.- MICHAELS, MITTMAN, CARLSON
"A comparison of relational and Codasyl approaches to data-base management".
ACM Computing Survey - volume 8, number 1 march 1976

11.- JOACHIM, SCHMIDT
"Some high level language constructs for data of type relation".
Association for computing machinery - volume 2, number 3
September 1977

12.- HUITS
"Requirements for languages in data-base systems".
ACM Douqué and g.m. Nyssen - 1975